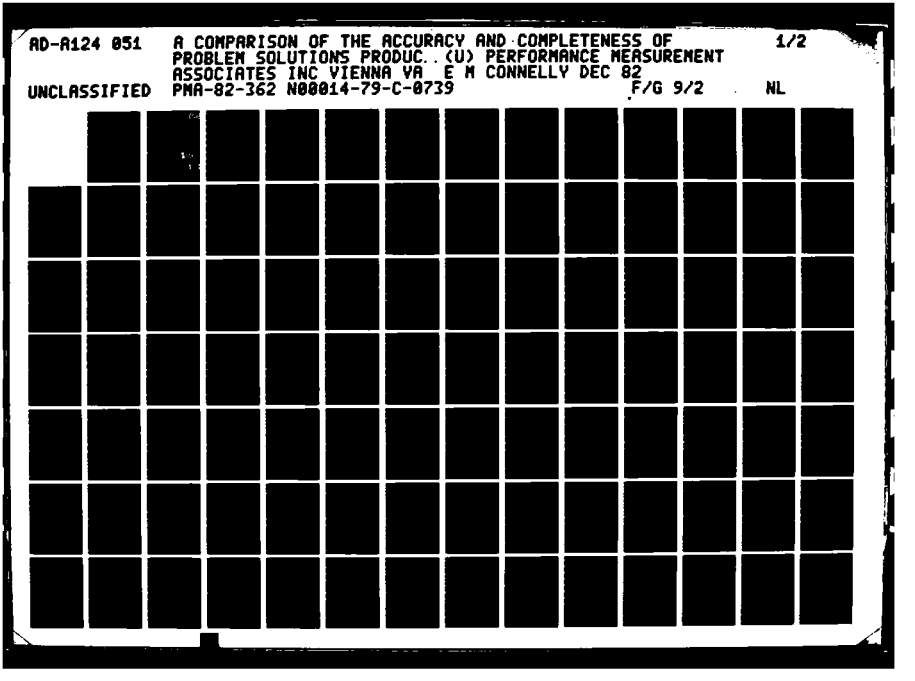
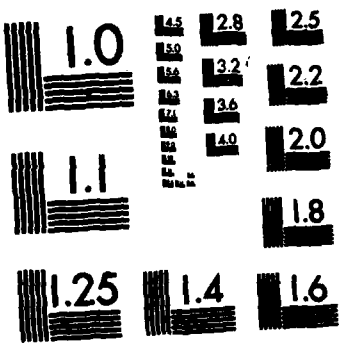

AD-A124 051 A COMPARISON OF THE ACCURACY AND COMPLETENESS OF 1/2
PROBLEM SOLUTIONS PRODUC.. (U) PERFORMANCE MEASUREMENT
ASSOCIATES INC VIENNA VA E M CONNELLY DEC 82
UNCLASSIFIED PMA-82-362 N00014-79-C-0739 F/G 9/2 NL



AD A124051

DTIC FILE COPY

(12)

TECHNICAL REPORT

A COMPARISON OF THE ACCURACY AND COMPLETENESS OF PROBLEM SOLUTIONS PRODUCED BY EXAMPLE-SOLUTIONS AND PROGRAM CODE

PMA

015

TECHNICAL REPORT

Edward M. Connelly

A COMPARISON OF THE ACCURACY AND COMPLETENESS OF PROBLEM SOLUTIONS PRODUCED BY EXAMPLE-SOLUTIONS AND PROGRAM CODE

This research is supported by Engineering Psychology Group, Office of Naval Research

December 1982

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER 82-362	2. GOVT ACCESSION NO. AD-A124051	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A COMPARISON OF THE ACCURACY AND COMPLETENESS OF PROBLEM SOLUTIONS PRODUCED BY EXAMPLE-SOLUTIONS AND PROGRAM CODE		5. TYPE OF REPORT & PERIOD COVERED Technical Report 1 Sept 79 - 28 Feb 82 6. PERFORMING ORG. REPORT NUMBER
7 AUTHOR(s) Edward M. Connelly		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0739
9 PERFORMING ORGANIZATION NAME AND ADDRESS Performance Measurement Associates, Inc. 410 Pine Street, S.E. Vienna, Virginia 22180		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N 42; RR 042-09 RR 042-09-01; NR 196-161
11 CONTROLLING OFFICE NAME AND ADDRESS Engineering Psychology Group Office of Naval Research 800 N. Quincy Street, Arlington, VA 22217		12. REPORT DATE December 1982 13. NUMBER OF PAGES 147
14 MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Department of the Navy Office of Naval Research Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16 DISTRIBUTION STATEMENT (of this Report) Approved for Public Release. Distribution Unlimited.		
17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18 SUPPLEMENTARY NOTES None		
19 KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Development, Programmer Behavior, Automatic Programming, Automatic Processor		

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

Use of an inductive processor to convert example-solutions into implied logic resulted in errors of omission no different from that which occurs with use of program code; however, such a technique provided a significant reduction in errors of commission. Feedback-aids in conjunction with example-solutions enabled the programmer to

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

engage in more complex problems with few errors of commission; those aids, to be most useful, need to include the implied logic. The feedback-aids that were optimal for the initial example-solutions were not suitable for the revision of incorrect example-solutions. In the population of programmers who participated in these experiments, the number of programming languages and the number of operating systems that the individual knew were established as the best predictors of success in developing example-solutions, as well as for writing program code.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTD)

TABLE OF CONTENTS (CONTD)

TABLE OF CONTENTS (CONTD)

TABLE OF CONTENTS (CONTD)

LIST OF FIGURES

LIST OF TABLES

LIST OF TABLES (CONTINUED)

LIST OF TABLES (CONTINUED)

LIST OF TABLES (CONTINUED)

ABSTRACT

Use of an inductive processor to convert example-solutions into implied logic resulted in errors-of-omission no different from that which occurs with use of program code; however, such a technique provided a significant reduction in errors-of-commission. Feedback-aids in conjunction with example-solutions enabled the programmer to engage in more complex problems with few errors of commission; those aids, to be most useful, need to include the implied logic. The feedback-aids that were optimal for the initial example-solutions were not suitable for the revision of incorrect example-solutions. In the population of programmers who participated in these experiments, the number of programming languages and the number of operating systems that the individual knew were established as the best predictors of success in developing example-solutions, as well as for writing program code.

INTRODUCTION

The software science literature is replete with descriptions of "improved" products on all levels: requirements-specification aids, programming languages, programming tools, design methodologies, and test strategies that are designed not only to improve the speed of producing computer programs, but also to improve their correctness and completeness. A similar statement can be made for systems designed to aid the computer user, including users of C^3 systems. Typically, the "improved" product is designed to automate a process previously performed by a user and, thus, to improve overall performance by simplifying the user's task. While some features of the new systems, taken out of the context of the other parts of the system, may appear to be useful, those same features may be not helpful or may be even harmful in the total system. For instance, new programming languages are developed with the goal of automating certain functions, resulting in a "simplier" system which is intended to be easier to use. But system simplicity by that measure alone is not specific enough to insure that the user can work more rapidly and with fewer errors. More detailed specifications of the useful features of a system based on experimental results are necessary in order to provide performance-based data for designing new systems.

An essential step in the development of a correct computer program is the correct and complete specification of the problem-solution to be implemented by the program. Numerous software engineering papers and texts point out that errors in specification of the problem-solution are propagated throughout the subsequent program development steps and, as a result, are difficult to detect and expensive to correct - if, in fact, the errors are detected. As discussed in a related technical report (Connelly, Johnson, Comeau, 1981), considerable efforts have been made in software engineering, design methodologies, programming languages, automatic programming, and program-test and validation procedures to provide aids for improved programs. While these efforts have produced aids and procedures which generally have resulted in improved programs, a serious difficulty in developing

software aids is that, before a particular aid has been developed, its value is unknown. This value can only be assessed after the aid has been developed and after users have become proficient in handling it. Thus aid-development becomes largely a "cut and try" process where aid designers attempt to produce solutions within their own area(s) of expertise.

But aids are frequently developed without the benefit of data regarding how well a human user will be able to perform with the aids. A fundamental problem in software development in general and specification of accurate and complete solutions specifications in particular is how to measure the ability of an individual to correctly accomplish each step of the task assigned to him. For instance, we should like to know for each task step the likelyhood of each type of error and the time required to complete the task. Further, we should like to know the effect of problem-complexity and certain aid-types on error-rates and completion-times. With this information, aid designers should be able to make informed choices regarding the design of interface-functions and the specification of machine and user tasks.

The research reported here is the second phase of a series of experiments to investigate the ability of individuals to correctly and accurately specify problem-solutions when working with various aid-designs at various levels of problem-complexity. The first phase of the research was reported in Connelly, Johnson, Comeau (1981), in which Experiments 1 and 2 were described and their results evaluated. Specifically, Experiments 1 and 2 investigated the ability of expert programmers and bookkeepers/accountants who were not expert programmers to develop example-solutions for a hypothetical Navy task force problem. The ability of the participants to develop example-solutions was evaluated as a function of the participants' background and experience, the complexity of the problem to be solved, and the level of processing provided by the computer. The problems used in the experiments reported herewere identical to those used in Experiments 1 and 2.

Several results of the first two experiments were used in the subsequent experiments. First, as expected, more errors occurred during work on the more complex problems.*

*Problem complexity and generalization of data are defined in a subsequent section titled, "Design of the Experiment".

However, the processing level, or generalization, of the example-solutions was found to be an important error-reducing factor, i.e., a significant reduction in errors occurred when data from example-solutions were processed into a standard form and presented to the participant.

A second result, and perhaps the most important, was that participants in both categories who performed well tended to use a systematic, step-by-step strategy in selecting example-solutions. This result, together with the first, noted above, suggested that feedback-aids might be designed to encourage participants to use a systematic strategy-by processing their example-solutions and feeding-back the resultant data to suggest possible additional inputs.

A third result of the first two experiments used in the subsequent experiments was that the number of years of advanced education (i.e., beyond high school) and the number of years of professional experience were found to be unimportant factors in predicting performance. As a consequence of this result, additional demographic factors were evaluated for the participants in the subsequent experiments in an effort to find important demographic predictors of performance.

The fourth result used in the subsequent experiments was the observation that only a few errors-of-commission occurred during the generation of the example-solutions. The majority of the errors that did occur were errors-of-omission. This intrigueing result influenced the design of Experiment 6, in which FORTRAN IV code was written to solve the same problems used in Experiment 1, so that a comparison of error-rates would be possible.

OVERVIEW OF EXPERIMENTS

The results of four experiments are reported here. Experiments 3 and 4 were designed to investigate the ability of expert programmers and non-programmers to develop accurate and complete example-solutions using various feedback aids at various levels of problem complexity. Experiment 5 investigated the capability of expert programmers using these feedback-aids to revise solution-specifications in which various numbers of initially incorrect entries had been introduced.

Finally, Experiment 6 called upon expert programmers to develop computer code written in FORTRAN IV for various levels of problem-complexity and various levels of data-input — a design intended to be analogous to that used in Experiment 1.

EXPERIMENTS 3 AND 4

Purpose

The purpose of Experiments 3 and 4 was to investigate the capabilities of programmers and non-programmers to develop example-solutions for problems using a set of problem-solving aids.

The problems solved were the same as those used in the previous experiments (1 & 2). Results of Experiments 1 and 2 show that the performance of both programmers and non-programmers was significantly influenced by the use of systematic, step-by-step strategies. The participants using systematic strategies were able to generate more complete example-solutions than were participants who did not use such strategies. As a result, the experimental feedback aids for Experiments 3 and 4 were designed to assist participants in developing a systematic strategy.

Method

The two participant categories were expert-programmers and experienced bookkeepers/accountants who were not expert programmers. The selection criteria for establishing whether a programmer was considered to be an expert-programmer included work with multiple languages, production of at least 10,000 lines of code, and experience with multiple computer systems. Selection criteria for the non-programming category (bookkeeper/accountant) included four years of schooling or experience in the bookkeeping/accounting field and not being an expert-programmer according to the criteria for programmers given above.

Participants were obtained by commercial personnel organizations which specialize in placing programmers and bookkeepers/accountants in temporary/permanent positions. Initial screening of the participants was done by the personnel agencies in accordance with the selection criteria provided to them. In addition, due to a low flow rate of participants,

particularly in the bookkeeping/accounting area, an advertisment was placed in a local newspaper stating that an evaluation of computer equipment was being conducted.

Participants who were obtained through agencies were paid in accordance with the policies of the agencies that obtained them. The programmers and bookkeepers/accountants who were obtained through the newspaper ad were paid $13.00 and $7.50 per hour, respectively.

Design of the Experiment

The experiment used a repeated-measures Latin Square design (Plan #9 cited in Winer, 1971, pp. 727-736). The factors investigated were:

1. 3 levels of problem complexity, where each level required a different amount of effort to correctly specify a problem solution, as measured by Halstead's E Metric (See Connelly, et al., 1981).

2. 3 levels of feedback aids, where each level presented different amounts of information to be considered by the participant.

3. 2 participant populations: experienced bookkeepers/accountants and expert computer-programmers.

Each factor was fixed, but the groups (G) and participants within the groups were random factors. The order of presentation for each group was randomized.

In Experiments 1 and 2, three processor levels were used. These processor levels were a function of the processor generalization capability. The three processor levels were:

Level B_1: The MIN and MAX transit-and stationing-times of input-examples were applied to all ships regardless of type or combination.

Level B_2: The MIN and MAX transit-and stationing-times of input examples were applied to all ships in each particular combination.

Level B_3: The MIN and MAX transit-and stationing-times of input examples were applied only to each ship type in each combination.

For Experiments 3 and 4, the problem complexity levels were varied, but the processor level was held constant at Level B_2 for all problems.

A pre-test problem was given to every candidate-participant. It consisted of a low-complexity problem requiring examples of three different ship-combinations combined with Feedback Level F_2 (to be described subsequently). When the pre-test was completed, the results were evaluated. If the participant input all three correct ship-combinations, and, in addition, the minimum and maximum times for transiting were not equal, and likewise for stationing, the participants were permitted to continue with the experiment-problems. Otherwise, the candidate-participants were not permitted to enter the experiment.

Procedure

Participants were scheduled for either a morning session beginning at 8:00 a.m. or an afternoon session beginning at 1:00 p.m. When a participant arrived, he/she was offered a refreshment and asked to fill out a biographical questionnaire. The questionnaire verified that the participant's experience satisfied the experiment entrance-criteria and obtained additional information regarding the level of the participant's experience in his/her particular field. If the participant's experience did not satisfy the criteria, he/she was not used in the experiment. If the participant's background was acceptable, the purpose of the experiment was explained, and the participant was provided with a consent form, having been assured that no personal risk was involved. The participant then signed the form to indicate that he/she understood these arrangements.

The participant was next seated in the experiment room. The room was approximately 12 x 16 feet in size, with a video tape recorder and video monitor located on one table and the computer and terminal on a separate table. Participants were asked to make themselves comfortable and to adjust the light and ventilation to their satisfaction.

Instructions for the experiment were presented in two parts, both of which were on video tape. The first part described the experiment-problem and gave a method for solving the problem, including an example-solution. The second part gave instructions on how to enter data into the computer and included an illustrative problem-solution. Since this portion of the instructions employed a dynamic display of the operation of the computer, it cannot be presented here.

After the instructions were presented, the participant was seated in front of the computer terminal. He/she was asked to use the manual key pad (consisting of keys labled 0-9 and ENTER). The participant was asked to enter example-solutions using numbers that corresponded to a ship-list containing various types of ships and their transiting and stationing times. The participants could refer to this list at any time during the experiment. In addition, a pad of paper and pencils were provided for notes, calculations, etc. These sheets were kept in each participant's file for reference.

Participants were told that up to one hour was allotted for each problem, and that the computer would automatically stop the problem when the hour was up. Participants were permitted to take a short break between test-problems if they desired.

The Task

The experiment task required specification of a logic for the selection of a hypothetical Navy task force. The task involved choosing ships from a ship-list which identified the ship type, the transiting time (i.e., time required for the ship to get from its present position to the desired site), and stationing time (i.e., the number of days the ship could remain on station with available provisions). The participant was required to specify example-solutions, i.e., example ship-combinations, for each problem worked. To accomplish this, in addition to specifying ship types, the participant also had to specify, through the example-solutions, the range of transiting and stationing times required. For instance, if the required transiting time was 10 days or less, the participant had to search the ship-list to find one example ship of an acceptable type that would satisfy the upper limit (10 days) for transiting time. To establish the lower limit for the range of acceptable

transiting times, the participant had to search the list to find another ship of an acceptable type with the lowest possible value for transiting time. Thus, the participant's task was to form example-solutions which not only contained proper combinations of ship types but also established the desired range of transiting and stationing times.

To conceal their relative complexities, the experiment problems were assigned random-number designations. Thus, Problem #93, shown in Figure 1, was the pre-test problem, in which there were only 3 possible combinations of ship types. In Problems #15, #52, and #31, shown in Figures 2, 3, and 4, there were 6, 9, and 13 possible combinations of ship types, respectively.

In the context of this report, Problem Level A_1, which denotes the easiest problem-task, corresponds to Problem #15 as presented in the experiment. Similarly, Problem Level A_2 corresponds to Problem #52 as presented in the experiment, and Problem Level A_3 corresponds to Problem #31.

Feedback Aids

The participants had various feedback-aids to use. It should be understood that the aids did not make use of any knowledge of the "correct solution" -- in fact, no information regarding the correct solution to each problem was entered into the computer for the experiment trials.

Feedback-Aid #1, demonstrated in Table 1, provided a display of the ship selection logic (SSL) implied by the participant's example solutions. It included: the ship types, MIN/MAX for transiting times, and MIN/MAX for stationing times for each ship-combination.

Feedback-Aid #2, demonstrated in Table 2, presented the participant's solutions ordered according to ship type. Each row shows a combination previously entered by the participant. The numbers along each row indicate how many ships of each type were in that combination. This aid was intended to help the participant to generate all the required combinations by systematically organizing those previously entered.

Now suppose the mission has been modified, you are asked to develop the ship selection logic (for the SSL subroutine) for the modified mission. The selection criteria for the task force have been modified to:

1. The ships needed for the task force are:
   - 1 Attack Aircraft Carrier (CVA or CVAN),

     And
   - 2 Submarines (SS or SSN),

     And
   - 4 Destroyers (DD),

     And
   - 1 Oiler (AO)

You must develop logic for each ship combination, as well as transit and stationing time logic specified by the mission type. Enter the logic statements for the SSL subroutine into the computer. Please start now.

Figure 2. Test Problem #15

Now suppose the mission has been modified. You are asked to develop the ship selection logic (for the SSL subroutine) for the modified mission. The selection criteria for the task force have been modified to:

1. The ships needed for the task force are:
   - 1 Attack Aircraft Carrier with Nuclear Propulsion (CVAN),

     And
   - 2 Guided Missile Cruisers (CG or CGN),

     And
   - 2 Submarines (SS or SSN),

     And
   - 3 Destroyers (DD),

     And
   - 2 Oilers (AO)

You must develop logic for each ship combination, as well as transit and stationing time logic specified by the mission type. Enter the logic statements for the SSL subroutine into the computer. Please start now.

Figure 3. Test Problem #52

Now suppose the mission has been modified. You are asked to develop the ship selection logic (for the SSL subroutine) for the modified mission. The selection cirteria for the task force have been modified to:

1. The ships needed for the task force are:
   - 2 Attack Aircraft Carriers (CVA) or
     1 Nuclear Attack Aircraft Carrier (CVAN),
     And
   - (2 Submarines (SS or SSN) and 3 Destroyers (DD)) or 3 Submarines (SS or SSN) and 2 Destroyers (DD)),
     And
   - 1 Oiler (AO).

You must develop logic for each ship combination,as well as transit and stationing time logic specified by the mission tkpe. Enter the logic statements for the SSL subroutine into the computer. Please start now.

Figure 4. Test Problem #31

Table 1

Example of Feedback-Aid #1
Ship Selection Logic (SSL)

Ship Type	No. of Ship Type	Transit Time MIN	Transit Time MAX	Stationing Time MIN	Stationing Time MAX
CVAN	0				
CVA	1	1	5	10	50
CA	0				
CGN	0				
CG	0				
DD	4	1	5	10	50
SSN	0				
SS	2	1	5	10	50
AO	1	1	5	10	50
TOTAL:	8				

CVAN Aircraft Carrier (Nuclear)
CVA Aircraft Carrier
CA Heavy Cruiser
CGN Guided Missle Cruiser (Nuclear)
CG Guided Missle Cruiser
DD Destroyer
SSN Submarine (Nuclear)
SS Submarine
AO Oiler

Table 2

Example of Feedback-Aid #2
Ships Ordered According to Ship Type

Number of Ships of Each Ship Type

	CVAN	CVA	CA	CGN	CG	DD	SSN	SS	AO
1.	1	0	0	2	0	3	0	2	2
2.	1	0	0	0	2	3	0	2	2
3.	1	0	0	1	1	3	0	2	2

CVAN	Aircraft Carrier (Nuclear)
CVA	Aircraft Carrier
CA	Heavy Cruiser
CGN	Guided Missile Cruiser (Nuclear)
CG	Guided Missile Cruiser
DD	Destroyer
SSN	Submarine (Nuclear)
SS	Submarine
AO	Oiler

Feedback-Aid #3 utilized an algorithm to form suggested next-logical combinations for the participants to consider. Table 3 shows an example of Feedback-Aid #3. Based on the computer's inducative interpretation of the participant's input examples, the computer identified incomplete ship-combination patterns and displayed suggested, further combinations for consideration. For instance, if the previous inputs had included the combinations 2 SS and 0 SSN, and 1 SS and 1 SSN, then the computer would suggest 0 SS and 2 SSN. Again, note that the suggested combinations developed by the computer were not based on any data regarding the correct solution -- the suggestions were made based only on the detection of incomplete combination-patterns in the participant's previous inputs.

Three Feedback-Aid Levels were formed for use in the experiment: F_1, which consisted of Aid #1, i.e., the SSL; F_2, which consisted of Aids #1 and #2, i.e., the SSL and the "ordered examples inputs"; and F_3, which consisted of Aids #1 and #3, i.e., the SSL and the "suggested combinations."

Data Entry

Participants used a numeric key pad (0-9 plus the ENTER key) to enter data into the computer. A participant could select from the following functions:

Function #1: Change the Page of Example Solutions. To change the example solution page on the terminal display, a participant would press Key #1 followed by the ENTER key. The computer would then request the page number that the participant wanted to view. The participant would enter the page number, resulting in a display of the desired page. (A page is a term referring to the information that is presented on a video monitor at a given time.) Additional pages were used to present additional information. In these experiments the pages contained the ship-numbers for each example-solution. Each page could display eight example-solutions and, thus, as each page became full, another page would be automatically assigned.

Function #2: Enter Example Solution. When the participant wanted to enter an example-solution, he/she pressed Key #2 and then the ENTER key. The computer then would request the number of ships in the example. The participant would respond with the appropriate number and then proceed to enter the ship ID numbers.

Table 3

Example of Feedback-Aid #3
"Next Suggested Combination"

Your previous inputs* have suggested the following ship combinations should be considered:

Number of Ships of Each Type

CVAN	CVA	CA	CGN	CG	DD	SSN	SS	AO
2	0	0	0	0	0	0	2	0
0	2	0	0	0	0	0	0	0

* Example Previous Input
1 Nuclear Carrier, 1 Aircraft Carrier, 2 Submarines

CVAN	Aircraft Carrier (Nuclear)
CVA	Aircraft Carrier
CA	Heavy Cruiser
CGN	Guided Missile Cruiser (Nuclear)
CG	Guided Missile Cruiser
DD	Destroyer
SSN	Submarine (Nuclear)
SS	Submarine
AO	Oiler

Function #3: Change Status of Example Solution. To change the status of any solution (ACCEPT or CANCEL), the participant would press Key #3 followed by the ENTER key. The computer would then request the number of the example-solution the participant wished to change. Pressing Key #3 changed the status of the example-solution from ACCEPT to CANCEL or from CANCEL to ACCEPT. As changes were entered, the computer updated the display to conform to the new ACCEPT/CANCEL status. Only accepted example-solutions were used in determining the computer's ship selection logic (SSL). A cancelled example-solution was ignored by the computer. Any example-solution could be cancelled by using the procedure described above. Also, a previously cancelled example-solution could be reintroduced at a later time by using the same procedure.

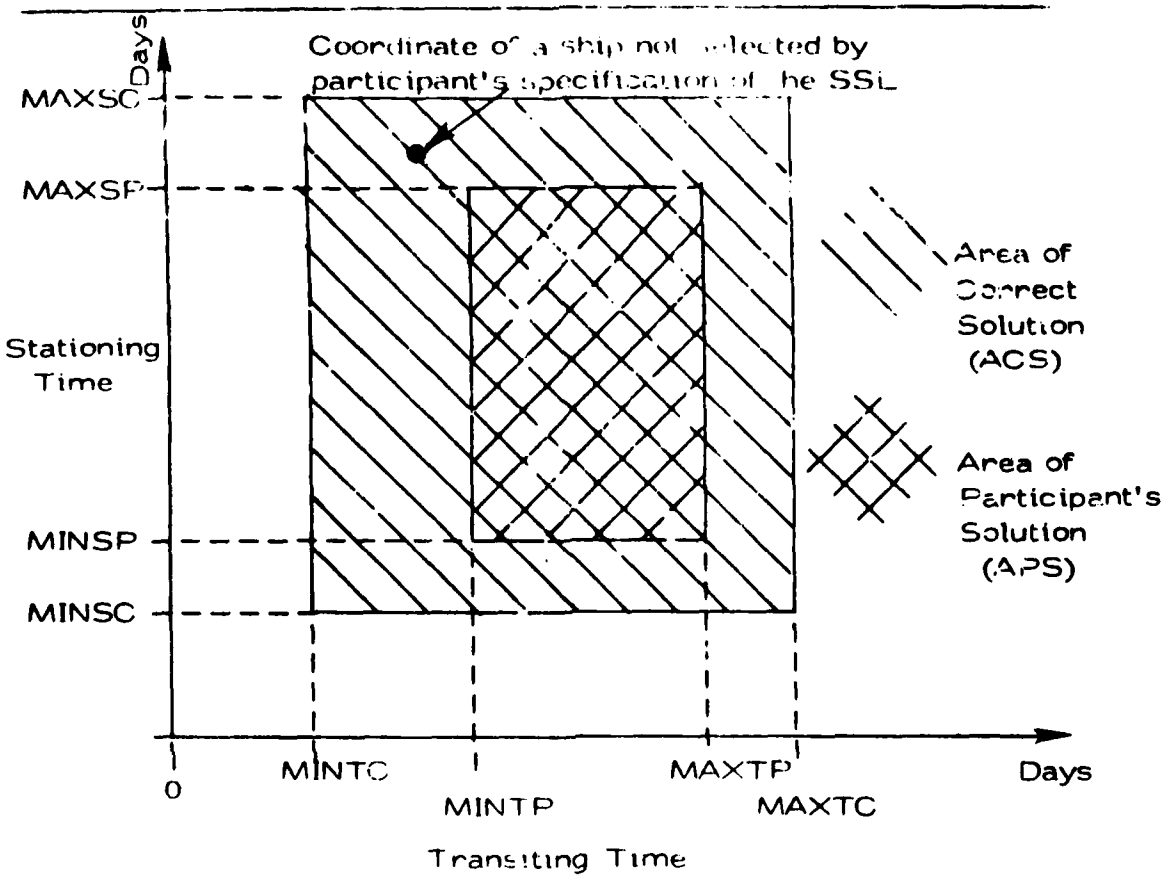
Function #4: View the Computer's SSL (Aid #1). To view the SSL formed by the computer (based on the example-solutions input by the participant), the participant pressed Key #5 followed by the ENTER key. Upon such a request, the computer would display the various ship-combinations that together formed the computer's SSL.

Function #6: View Aid #2 (when available). To view Aid #2, the participant pressed Key #6 followed by the ENTER key.

Function #7: View Aid #3 (when available). To view Aid #3, the participant pressed Key #7 followed by the ENTER key.

Performance Measures: Absolute and Relative Area Score

Absolute Area-Score, or the Probability of Correctly Selecting an Acceptable Ship: A summary performance-measure was developed to reveal how well each participant solved the test problems. The measure was constructed by scoring the MIN/MAX transiting and stationing times for each ship type. As shown in Figure 5, the range of correct values for the MIN/MAX transiting and stationing times, which was a function of the ships on the available ship list, can be viewed as an area in the problem-space. Every ship of a particular type whose coordinates fell within that area was acceptable. However, if a participant's ship-selection logic provided a minimum for either transiting or stationing time greater than the correct minimum, or if the maximum for either transiting or stationing was less than the



(ACS) Area of correct solution = (MAXTC - MINTC) (MAXSC - MINSC)

(APS) Area of participant's solution = (MAXTP - MINTP) (MAXSP - MINSP)

MINSP	——	Minimum Stationing - Participant
MINSC	——	Minimum Stationing - Correct Solution
MINTC	——	Minimum Transiting - Correct Solution
MINTP	——	Minimum Transiting - Participant
MAXSC	——	Maximum Stationing - Correct Solution
MAXSP	——	Maximum Stationing - Participant
MAXTP	——	Maximum Transiting - Participant
MAXTC	——	Maximum Transiting - Correct Solution

Figure 5. Method of Computing Probability of Acceptable Ship Selection

correct maximum then the "area" specified by the participant was contained entirely within the "correct area."

As a result, it was possible that a ship with acceptable coordinates might not be selected according to the SSL specified by a given participant. The probability that an eligible ship of a given type (denoted for convenience by the subscript i) would be selected by the participant's SSL was the ratio of the areas just described.

That is:

$$P_i = \frac{APS}{ACS} \qquad (1)$$

where: APS = Area of participant's SSL for ship type i.
ACS = Area of correct SSL for ship type i.

The probability of selecting all eligible ships for a correctly specified combination of ships is the product of the P_i over all the ship types in that combination.

That is:

$$PC_k = \prod P_i \text{ over all } i \text{ in combination } k \qquad (2)$$

where: C_k represents the k^{th} correctly specified ship-combination.

For a problem requiring N correct ship-combinations, the <u>average probability</u> of selecting an eligible ship regardless of type, which we shall call the "absolute area-score," or simply the "area-score," is given by:

$$\text{Area-Score} = \frac{1}{N} \sum_{k=1}^{N} PC_k \qquad (3)$$

Thus, the area-score reflects the probability that acceptable ships were in fact accepted by the SSL specified by the participant. If a particular ship combination was not specified by the participant, the corresponding PC_k value was zero. Note that the area-score did not carry any penalty for specifying incorrect ship combinations. Note further that the area-score was identical to Performance Measure 1 used in Experiments 1 and 2.*

*See Connelly, Comeau, & Johnson 1981.

Relative Area-Score: Area-score is a function of the experimental factors and the innate capability of each participant. A different measure, one that tends to reduce the effect of performance-variations due to variation in participant capability, is called the "relative area-score." It was defined for an experiment problem as the area-score for that problem minus the area-score for the participant's pretest problem.

Thus: Relative Area-Score = Area Score on Problem - Area-Score on Pretest Problem (4)

Procedure Measure

Two participant strategy measures were developed to test the relationship between the strategy used by participants in developing example-solutions and their resulting performance-scores. One strategy-measure, termed a procedure-measure, was designed to examine the pattern of choices among the options available to participants working with the computer. Referring to Figure 6, it can be seen that once a participant entered an example-solution into the computer he/she had three choices for the next step. One choice was to input another example-solution. Another choice was to view Aid #1, the computer ship-selection logic (SSL), to determine the effect of all example-solutions entered up to that time. The third possible choice was to view another Aid, #2 or #3, if available. The relative frequency of these three choices is represented by the Probabilities P_1, P_2, and P_3, respectively.

The value of P_1 reflects the propensity of the participant to input either a single example-solution or multiple example-solutions in sequence. If a participant's P_1 has a low value - near 0 - it may be concluded that the participant tends primarily to input a single example-solution. Such a participant would typically request a display of the SSL or another aid after each input. If P_1 is very high - near 1.0 - then the value indicates that multiple examples were input before proceeding to one of the other input states. In fact, the value of P_1 provides an estimate of the mean number of times example-solutions are input in sequence, as follows:

$$\text{Mean number of entries in sequence} = \frac{1}{1 - P_1}. \quad (5)$$

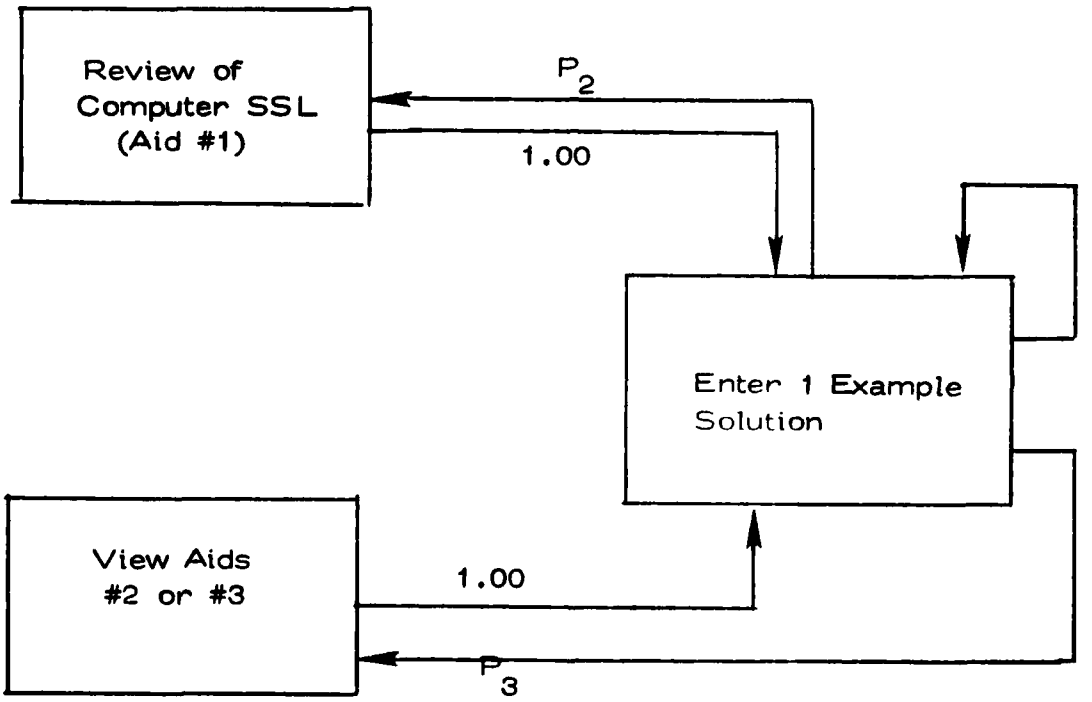


Figure 6. Participant Procedural-Strategy

In a similar way the values of P_2 and P_3 reflect the propensity of a participant to view the SSL or other aids. A large value of P_2, for instance, indicates a strategy of inputting a single (or only a few) example-solutions and then reviewing the effect on the SSL. A large value of P_3 indicates a propensity to input one (or only a few) example-solutions and then to view Aid #2 or #3. When these aids are unavailable, i.e., at Feedback Aid Level F_1, P_3 equals 0.

Since the three alternatives represent all of the available choices, the sum $P_1 + P_2 + P_3$ will always equal 1.0. Thus, as P_1 increases, the sum $P_2 + P_3$ will automatically decrease. As a result, the three variables are correlated, and the three cannot be used together in a regression as independent variables to investigate the relationship between participant-strategy and task-performance. But, one probability or possibly two, if these were shown to be independent, could be used as independent variables.

Combinational Measure (CM)

Another measure of participant strategy involved the nature of the relationship between one example-solution and the previous solutions input by a participant. In each problem logical OR conditions were specified which forced a participant to develop several different ship combinations in order to completely specify an SSL. Many of these combinations required variation within one ship type - submarine, for example, either nuclear (SSN) or non-nuclear (SS) - while others required variation over more than one type.

If two submarines (SS or SSN) were specified in conjunction with other ship types, then each specific combination of the other ships had to be associated with three possible variations of SSN's and SS's, as follows:

2 SS and (other ships)

OR

1 SS and 1 SSN and (other ships)

OR

2 SSN and (other ships)

Similarly, if 3 submarines were specified, then each specific combination of other ships had to be associated with four variations of SS's and SSN's, as follows:

3 SS and (other ships)

OR

2 SS and 1 SSN and (other ships)

OR

1 SS and 2 SSN and (other ships)

OR

3 SSN and (other ships).

Often, variations (i.e., "OR" conditions) within one ship-type (e.g., submarine) were specified in combination with variations (OR conditions) within one or more of another type of ship (e.g., nuclear or non-nuclear aircraft carrier). Variations over multiple ship-types required the generation of numerous combinations, and it was difficult to generate all such combinations in the absence of a systematic method.

The combinational strategy-measure was designed to detect whether a participant tended to construct example-solutions systematically - i.e., by effecting changes only within one ship-type (e.g., SS or SSN) in successive example-solutions and by providing all such required variations within one ship-type in sequence before proceeding to the next type.

The measure was computed as follows:

1. If on two successive example-solutions more than one ship type was changed - score 0.

2. If on two successive example-solutions changes occured only within one ship type - score 1.

3. If on three successive example-solutions changes occured only within one ship type - score 1 for the first change (according to condition 2 above) and 2 for the second change.

4. And so forth for 4 or more successive example-solutions.

Data Analysis

Analysis 1: ANOVA

Purpose. The purpose of this analysis was to determine whether the main experiment variables (problem-complexity, feedback-aids, and participant-population) affected overall performance.

Method. An ANOVA was used to evaluate main and interactive effects.

Results. Table 4 shows that participant population and problem-complexity had a significant effect on performance, as measured by the area-score, but that feedback-aids did not.

Analysis 2: Means

Purpose. To investigate the significance of the differences in experimental-cell mean scores.

Method. An a posteriori multiple comparison of means test (Student-Newman-Keuls (SNK)) was used to determine the statistical significance of treatment-means.

Results. Table 5 contains the mean scores for each of the experiment cells. As shown (and as expected from the previous experiments) programmers were better than bookkeepers/accountants, but not always substantially better. For instance, in cell A_1F_1 performance of both programmers and bookkeepers/accountants was quite close, though in the other cells the performance was substantially different. For programmers, cell-performance ranges from a high of .778 to a low of .525; bookkeepers/accountants' cell-performance ranges from .589 to .175.

The results of the SNK test for programmers are shown in Table 6, where a statistically significant ($P \leq .05$) difference was found between performance in cell A_1F_2, where superior performance was obtained, and cell A_3F_3, where the lowest performance was obtained.

Table 4

Analysis of Variance Table

ANOVA Table Follows:

Source of Variation	SS	DF	MS	F	P≤
Between Subjects	26.6563263	59			
C	3.7321777	1	3.7321777	9.303	.005
Rows	0.1865654	2	0.0932827	0.233	NS
C X Rows	1.0740433	2	0.5370216	1.339	NS
Sub within Group	21.6635399	54	0.4011767		
Within Subjects	3.0878906	120			
A	0.4542351	2	0.2271175	10.341	.001
B	0.0535393	2	0.0267696	1.218	NS
AC	0.0353241	2	0.0176620	.804	NS
BC	0.0102844	2	0.0051422	.234	NS
AB	0.0600395	2	0.0300198	1.366	NS
ABC	0.1024704	2	0.0512352	2.33	NS
Error (Within)	2.3719132	108	.0219621		

A - Problem-Complexity
B - Feedback-Aids
C - Participant-Categories (Programmers, Bookkeepers/Acct)

Table 5

Mean Scores for Both Participant-Categories

Programmers

	A_1	A_2	A_3
F_1	.593	.678	.686
F_2	.778	.595	.656
F_3	.745	.627	.525

Bookkeeper/Accountants

	A_1	A_2	A_3
F_1	.589	.391	.175
F_2	.305	.422	.392
F_3	.464	.217	.337

A_1, A_2, A_3 Problem-Complexity Levels

F_1, F_2, F_3 Level of Feedback-Aids

Table 6

Student - Newman - Keuls Test

Results for Programmers

Problem Level / Feedback Aid		A_3F_3	A_1F_1	A_2F_2	A_2F_3	A_3F_2	A_2F_1	A_3F_1	A_1F_3	A_1F_2
	Means*	.525	.593	.595	.627	.656	.678	.686	.745	.778
A_3F_3	.525	-								
A_1F_1	.593	.068	-							
A_2F_2	.595	.070	.002	-						
A_2F_3	.627	.102	.034	.032	-					
A_3F_2	.656	.131	.063	.061	.029	-				
A_2F_1	.678	.153	.085	.083	.051	.022	-			
A_3F_1	.686	.161	.093	.091	.059	.030	.008	-		
A_1F_3	.745	.220	.152	.150	.118	.089	.067	.059	-	
A_1F_2	.778	** .253	.185	.183	.151	.122	.100	.092	.033	-
	LSR	9	8	7	6	5	4	3	2	1

*Means of all programmers' performance in each cell, i.e., A_1F_1, A_1F_2, etc.
**Indicates row is significantly different from column value $p \leq .05$.

A_1, A_2, A_3 Problem-Complexity Levels
F_1, F_2, F_3 Level of Feedback-Aid

Each matrix entry = row value - column value
(e.g., entry in second row, first column is .068 = .593 - .525)

A more complex picture is shown in Table 7, which gives the results of the SNK test for bookkeepers/accountants. Performance in cell A_1F_1 was shown to be significantly superior to performance in all other cells except A_1F_3. And performance in A_1F_3, in turn, was shown to be significantly superior to performance in A_3F_1 and A_2F_3.

Analysis 3: Percentage of Participants with Perfect Score

Purpose. To identify the percent of subjects with perfect scores and, thus, to reveal the impact of experimental factors on superior performers.

Method. Calculate the percent of participants achieving a 1.0 score.

Results. The results for programmers, given in Table 8, show that cell A_1F_2 had the highest percentage, 60% of the participants who achieved perfect scores. In contrast, only 20% of the participants achieved perfect scores in A_3F_3. This reflects the same trend as for the average scores reported in the previous analysis and gives some evidence that feedback-level F_2 may have been more useful to superior performers than feedback-level F_3. And, apparently, experimental factors affected superior performers as well as less-than-superior performers.

But a somewhat different story was presented by the results for bookkeepers/accountants, also shown in Table 8, where approximately 40% of the participants achieved perfect scores for cell A_1F_3, a percentage exceeding that of cell A_1F_1. Also, the percent of performers achieving a perfect score in cell A_1F_1 was equal to that in A_2F_2, which was again different from the results using average score described in the previous analysis. Apparently, then, for bookkeepers/accountants, the experiment factors affected superior performers in a different way then they affected the less-than-superior performers.

Figures 7 and 8 give the mean scores for each participant-category versus problem-complexity and feedback-levels, respectively. As shown in the Figure 7, increasing problem-complexity resulted in a reduction in average performance for both participant-categories. Feedback-levels in the order F_1, F_2, F_3 resulted in a decreasing level of average performance

Table 7

Student - Newman - Keuls Test

Results for Bookkeepers/Accountants

Problem Level / Feedback Aid		A_3F_1	A_2F_3	A_1F_2	A_3F_3	A_2F_1	A_3F_2	A_2F_2	A_1F_3	A_1F_1
	Means*	.175	.217	.305	.337	.391	.392	.422	.464	.589
A_3F_1	.175	-								
A_2F_3	.217	.042	-							
A_1F_2	.305	.130	.088	-						
A_3F_3	.337	.162	.120	.032	-					
A_2F_1	.391	.216	.174	.086	.054	-				
A_3F_2	.392	.217	.175	.087	.055	.001	-			
A_2F_2	.422	** .247	.205	.117	.085	.031	.030	-		
A_1F_3	.464	** .289	** .247	.159	.127	.073	.072	.042	-	
A_1F_1	.589	** .414	** .372	** .284	** .252	** .198	** .197	** .167	.125	-
	LSR	9	8	7	6	5	4	3	2	1

*Means of all bookkeepers/accountants' performance in each cell, i.e., A_1F_1, A_1F_2, etc.

**Indicates row is significantly different from column value $p \leq .05$.

A_1, A_2, A_3 Problem-Complexity Levels
F_1, F_2, F_3 Levels of Feedback-Aid
Each matrix entry = row value - column value
(e.g., entry in second row, first column is .042 = .217 - .175)

Table 8

Percentage of Perfect Scores for Both Participant-Groups

Programmers

	A_1	A_2	A_3
F_1	40	50	30
F_2	60	40	40
F_3	40	40	20

Bookkeeper/Accountants

	A_1	A_2	A_3
F_1	30	10	0
F_2	0	30	0
F_3	40	0	20

A_1, A_2, A_3 Problem-Complexity Levels

F_1, F_2, F_3 Level of Feedback-Aids

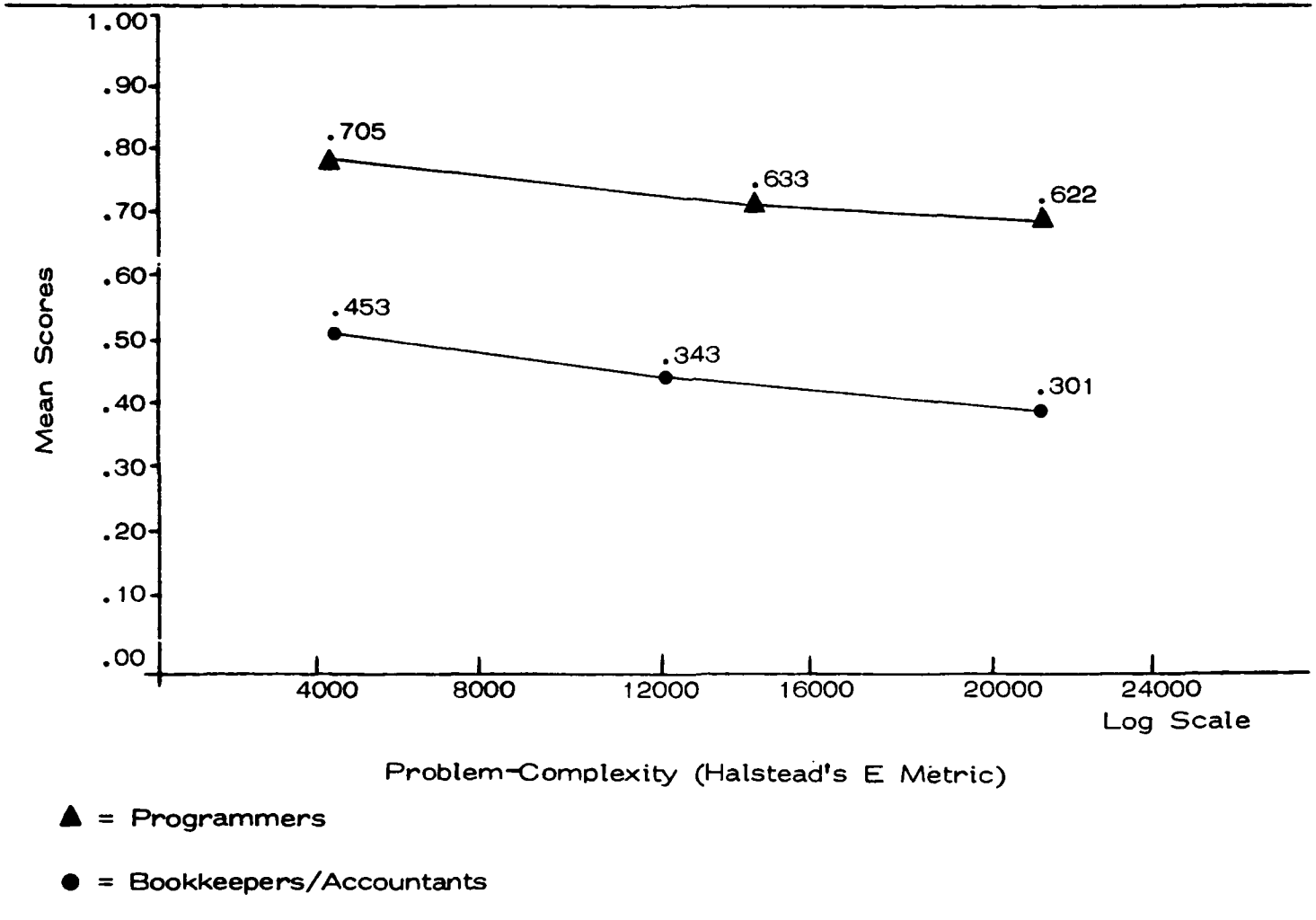
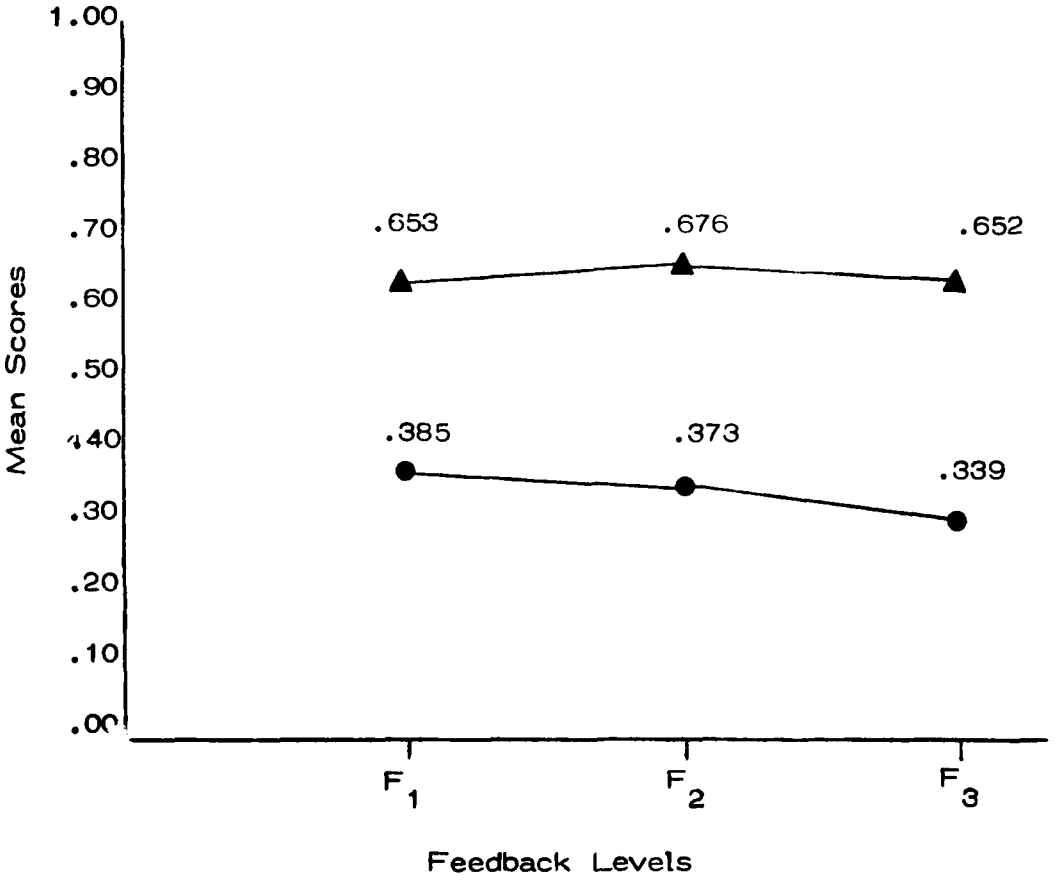


Figure 7. Mean Scores for each Participant Category vs. Problem Complexity



F_1= SSL

F_2= SSL & Ordered Example

F_3= SSL & Next Logical Solutions

▲ = Programmers

● = Bookkeepers/Accountants

Figure 8. Mean Scores for each Participant-Category vs. Feedback Levels

for bookkeepers/accountants. For programmers, however, there was an increase (but not a statistically significant increase) in average performance for level F_2 over that for F_1 and F_3.

Analysis 4: Effect of Feedback-Aids on Sub Populations and Relative Area-Score

Purpose. To determine the effect of the experimental variables on portions of the participant-populations as measured by the relative area-score. Rational for this analysis was that:

> Combinational strategy was known to be related to performance (from Experiments 1 and 2).* Feedback Aids #2 and #3 were designed based on that result and were used in Experiments 3 and 4 to assist participants in developing systematic (i.e., combinational) strategies. But, the analysis above of data from Experiments 3 and 4, using the absolute area-score, did not reveal any substantial benefit provided by the aids.

To further investigate the existence of aid impact, a set of analyses, labeled A through M was conducted. The purpose of these analyses was to determine whether the aids-effect was a function of:

1. Various combinations of feedback-aids,
2. Absolute vs. relative area-score,
3. All performers vs. performers who achieved less than a perfect score on the pretest problem, and
4. Various combinations of participant sub-populations.

Method. Correlations, and univariate and linear multivariate regressions were run to analyze the relationship between the feedback aids (and other independent variables) and area-score, both absolute and relative, for conditions 1 through 4 above. Rationale and details of the method for each condition are given in the following paragraphs.

Combinations of Feedback-Aids. Descriptions of the content of each of the three feedback-aids have been given previously. In order to determine performance (as measured by the area-score) for pairs of aids, correlations and regressions were run with various numerical codes assigned to each aid. When performance for all of the three aids was analyzed, the numerical codes were:

*Ibid.

Aid	Code
#1	1
#2	2
#3	3

The code was the numerical value assigned to the aid in the regression analyses.

When performance with Aids #1 and #2 alone was compared, the codes were:

Aid	Code
#1	1
#2	2
#3	Performance for Aid #3 removed from data file.

In the last case, when performance with Aids #1 and #2 was compared to that with Aid #3 (this condition is noted, in the subsequent Tables, as "1 and 2, 3"), the following numerical codes were used:

Aid	Code
#1	1
#2	1
#3	2

In this case, Aids #1 and #2 were rendered equivalent by the code.

Area-Score vs. Relative Area-Score. When performance was measured with an absolute measure, the area-score was used as the dependent variable. When the effect of the experiment-factors on a participant's area-score relative to that participant's score on the pretest was of interest, the relative area-score was used as the dependent variable. Since the pretest problem was a (relatively) easy problem, performed with feedback Aid #1, the relative area-score was taken to be a measure of the relative effect on performance of the other aids and the more difficult problems. In contrast, the absolute area-score included that relative effort plus the varying capabilities of each participant.

Participant Ability. Another condition varied in the analyses was the inclusion or exclusion of data for performers who achieved a perfect score of 1.0 on the pretest. The rational was that the feedback-aids may not have been required and therefore may not have been beneficial to individuals who used a systematic strategy independent of the aids, though the aids may still have helped those who would not have done well without them. A score of less than 1.0 on the pretest was taken as an indication of the possible need for an aid in the experiment problems.

Participant Sub-Population. Finally, analyses were performed for three participant-population conditions: both programmers and bookkeepers/accountants, programmers alone, and bookkeepers/accountants alone.

Results. Table 9 gives the results of correlation analyses A through M, indicating each analysis condition and the correlation of the feedback-aid with area-score or relative area-score.

In analyses A through H, area-score was correlated with various combinations of feedback-aid and participant sub-populations factors. While there was some variation among the correlation coefficients, the absolute values of the coefficients were small - almost zero. This reflected a lack of strong association between feedback-aids and performance as measured by area-score.

In analyses I through M, however, relative area-score was correlated with the various combinations of feedback-aid and participant sub-populations factors. A substantial increase in correlation-values was observed. It was noted that the correlation of the relative-score for non-programmers (i.e., bookkeepers/accountants) was higher than that for programmers. This may have been because non-programmers could use Aids #2 and #3 to greater benefit.

Further, a comparison of the results of analyses K and L, i.e., of performance with Aid #2 vs. Aid #3, showed that Aid #3 provided the higher correlation. This suggests that Aid #3 affected performance to a greater degree than did Aid #2.

Table 9

Correlation of Feedback-Aid vs.
Area-Score Under Various Conditions

Analysis	Feed-back Aid	Area Score	Superior Performance on Pre-test	Partici-pant Popula.	Correla. Coeff.
A	1,2,3	Absolute	Included	Both	-.039
B	1,2	Absolute	Included	Both	-.011
C	1,3	Absolute	Included	Both	-.049
D	2,3	Absolute	Included	Both	-.037
E	1&2,3	Absolute	Included	Both	-.040
F	1,2,3	Absolute	Included	Prog.	-.058
G	1,2,3	Absolute	Included	Non-Prog.	-.022
H	1,2,3	Absolute	Excluded	Both	-.031
I	1,2,3	Relative to Pre-Test	Included	Prog.	.373
J	1,2,3	"	Included	Non-Prog.	.442
K	1,2	"	Included	Both	.407
L	1,3	"	Included	Both	.452
M	1,3	"	Excluded	Both	.467

Finally, in Analysis M, which included only the sub-population of all participants who did not achieve a perfect 1.0 area-score on the pretest problem, increase in the correlation coefficient was obtained over the same analysis (L) conducted with the total participant population.

Table 10 gives the results of univariate regression analyses in which feedback-aid combinations were coded as described previously for the independent variables and relative area-score was the dependent variable. In terms of percent variance explained in analyses I and J, a greater percent of non-programmers' variance was explained than programmers'. Further, Feedback-Aid #3 (Analysis L) explained a greater percent of variance than did Aid #2 (Analysis K). And finally, 21.8% of the variance was explained by data involving Feedback-Aids #1 and #3 and the sub-population that did not achieve a perfect 1.0 on the pretest.

The effect of feedback-aids together with other factors on the relative area-score in a multivariate regression is given in Table 11. Results from Analysis H, which uses area-score, are presented as a reference to reemphasize the insignificant effect of feedback-aids and the significant effect of combinational-strategy. In Analyses I, J, and K, where relative area-score was used, the significant effect of combinational-strategy and the increased effect of feedback-aids were demonstrated.

Analysis 5: Effect of Session-Sequence

Purpose. To determine the effect of session-sequence on performance.

Method. A univariate analysis was conducted with session-sequence numbers as the independent variable and area-score as the dependent variable.

Results. The results of a univariate regression analysis, shown in Table 12, indicate that session sequence was a statistically significant factor ($P \leq .1$), but that it explained only 1% of the score variance. Apparently, session-sequence, while significant, did not affect performance in a substantial way.

Table 10

Correlation/Regression Analysis
Effect of Feedback-Aids on Relative Area-Score

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
Analysis I (Programmers)					
Feedback-Aid 1,2,3	.373	.0945	13.9	3.05	.002
Analysis J (Non Programmers)					
Feedback-Aid 1,2,3	.442	.1348	19.6	3.62	.001
Analysis K (All Participants)					
Feedback-Aid 1,2	.402	.1140	16.5	4.75	.001
Analysis L (All Participants)					
Feedback-Aid 1,3	.452	.1418	20.5	5.41	.001
Analysis M (All Participants except those scoring 1.0 on Pretest)					
Feedback-Aid 1,3	.467	.1507	21.8	4.47	.001

Table 11

Correlation Regression Analysis
Effect of Feedback-Aids on Area-Score
and Relative Area-Score

Independent Variable	Correla. Coeff.	Regre. Coeff	% Variance Explained	t-Value	P ≤
Analysis H (Area-score and all participants except those scoring 1.0 on pretest)					
Feedback-Aids 1,2,3	-.031	-.0326	42.6	-.525	NS
Combinational Strategy	.651	.7271		7.30	.001
Analysis I (Programmers relative area-score)					
Feedback-Aids 1,2,3	.373	.0960	15.6	3.10	.001
Combinational Strategy	-.113	-.0514		-1.06	NS
Analysis J (Bookkeepers/accountants relative area-score)					
Feedback-Aids 1,2,3	.442	.1298	26.7	3.61	.001
Combinational Strategy	.294	.1284		2.27	.05
Analysis K (All participants, relative area-score)					
Feedback-Aids 1,2	.407	.1106	17.1	4.51	.001
P_2	-.139	-.0301		-0.85	NS
Analysis L (All participants, relative area-score)					
Feedback-Aids 1,3	.452	.1288	26.1	4.91	.001
Problem Complexity	.189	.0000033		1.72	.05
Time	.295	.000026		1.91	.05
Analysis M (All participants except those scoring 1.0 on pretest, relative area-score)					
Feedback-Aids 1,3	.467	.1407	28.2	4.18	.001
Problem Complexity	.204	.0000043		1.75	.05
Time	.282	.000023		1.39	.10

Table 12

Correlation/Regression Analysis:
Session-Sequence Factor vs. Area-Score

Independent Variable	Correlation Coefficient	Regression Coefficient	% Variance Explained	t-value	P ≤
		Univariate Regression			
Session-Sequence Number	.106	.053	1.1	1.40	.10

Analysis 6: Demographic Factors and Performance

Purpose. To investigate the effect of demographic factors on participants' performance

Method. Univariate and multivariate regression analyses were used to evaluate the effects of age, years-of-higher-education, and years-of-experience on the overall performance.

Results. Figures 9 and 10 plot the mean score vs. the years-of-higher-education for programmers and bookkeepers/accountants, respectively. The scatter diagrams show a broad distribution of scores seemingly uncorrelated with years-of-higher-education. Similar scatter diagrams are shown in Figures 11 and 12, which plot the mean scores vs. years-of-experience for programmers/and bookkeepers/accountants, respectively. Again, the large spread of values suggests a low correlation between years-of-experience and mean score for both programmers and bookkeepers/accountants.

This result was verified for all participants by the univariate and multivariate regressions shown in Table 13. Here, age was the only factor providing a significant univariate regression. Age was negatively correlated with score. This was consistant with results found in Experiments 1 and 2.*

In a multi variate regression, three factors were found to have regression coefficients significantly different from zero. These were age, higher-education, and years-of-experience. Higher-education and years-of-experience, although positively correlated with overall area-score, had low correlation values, viz., .077 and .085, respectively. Further, these three variables, although they are usually the dominant factors considered in hiring, promoting, and establishing salary, together explained only about 15% of the area-score variance.

Table 14 gives the results of univariate and multivariate regression analyses in which demographic variables were the independent variables and combinational strategy score was the dependent variable. Results similar to those for area-score were found here: age, which was negatively correlated with the strategy score, was the only variable found to be statistically

*Ibid.

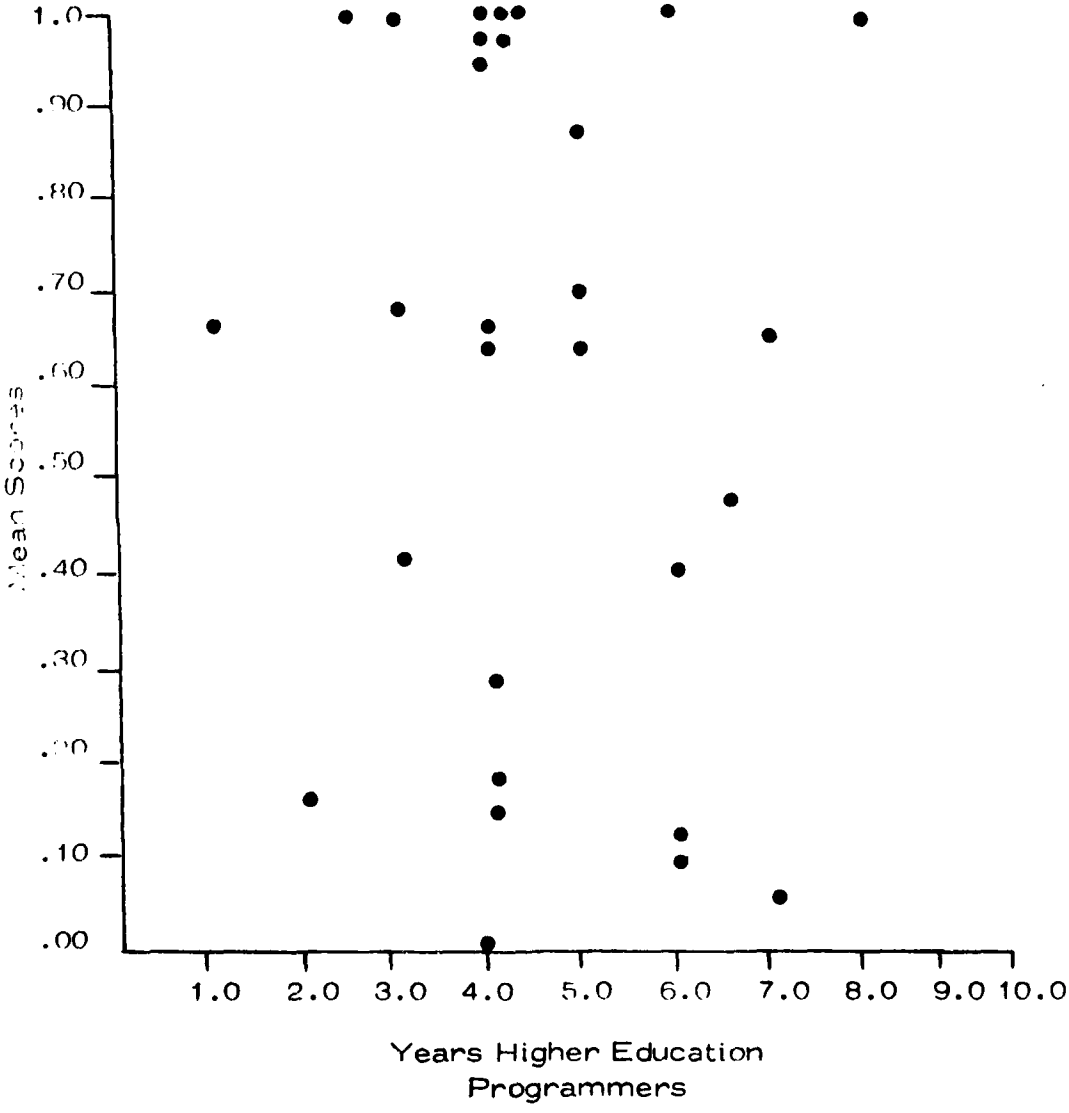


Figure 9. Mean Score vs. Years Higher Education: Programmers

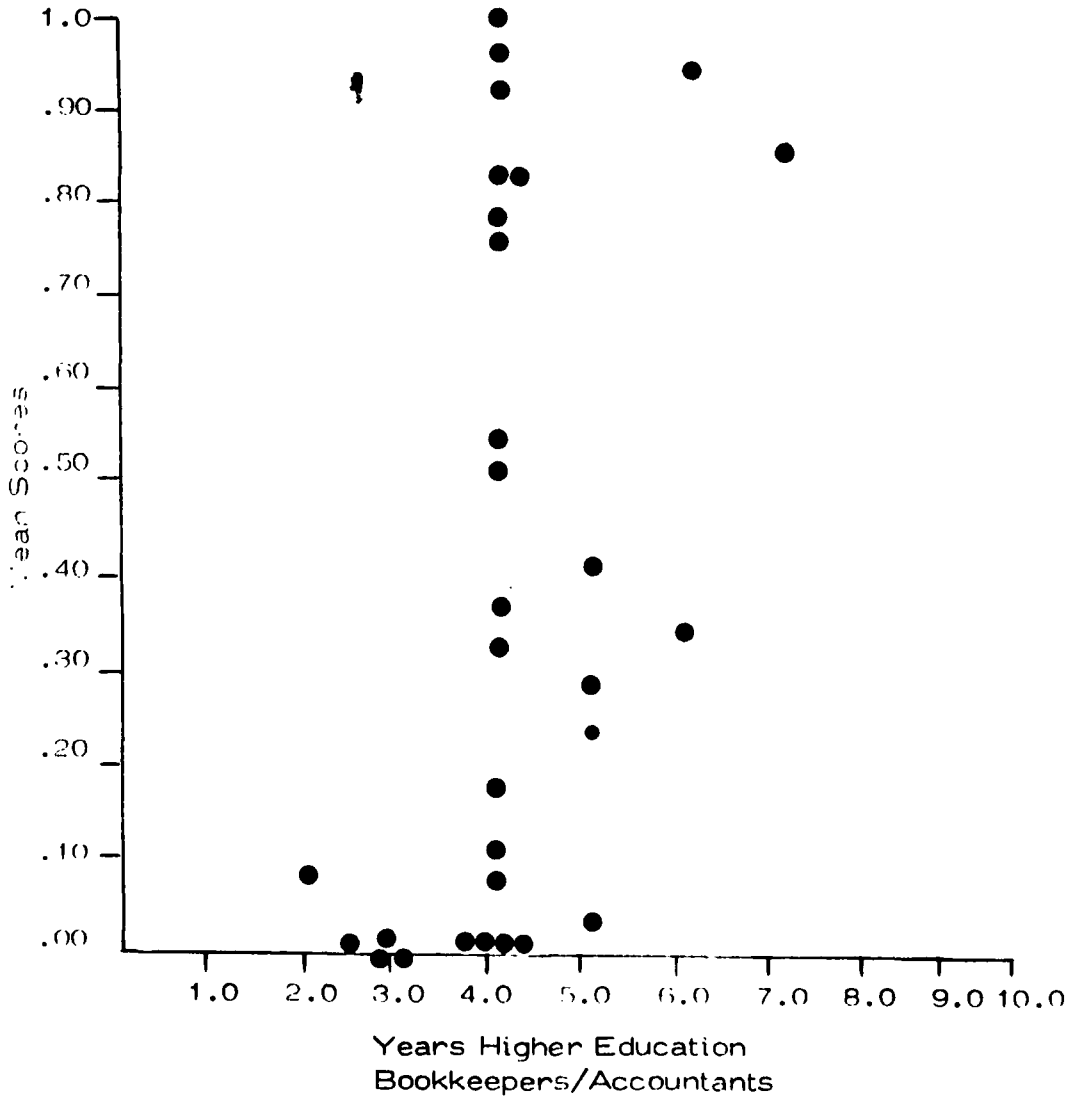


Figure 10. Mean Score vs. Years Higher Education: Bookkeepers/Accountants

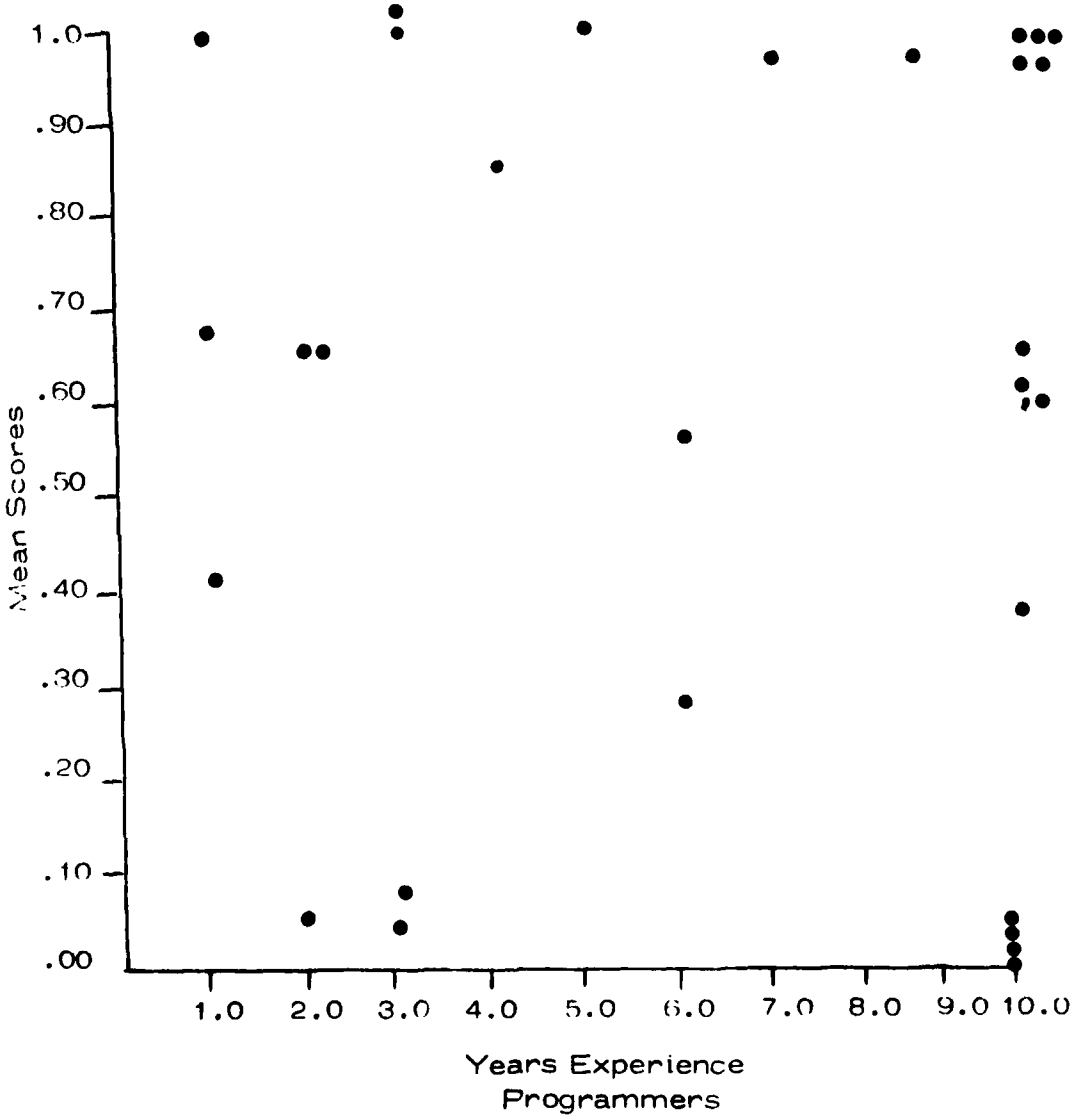


Figure 11. Mean Score vs.Years Experience: Programmers

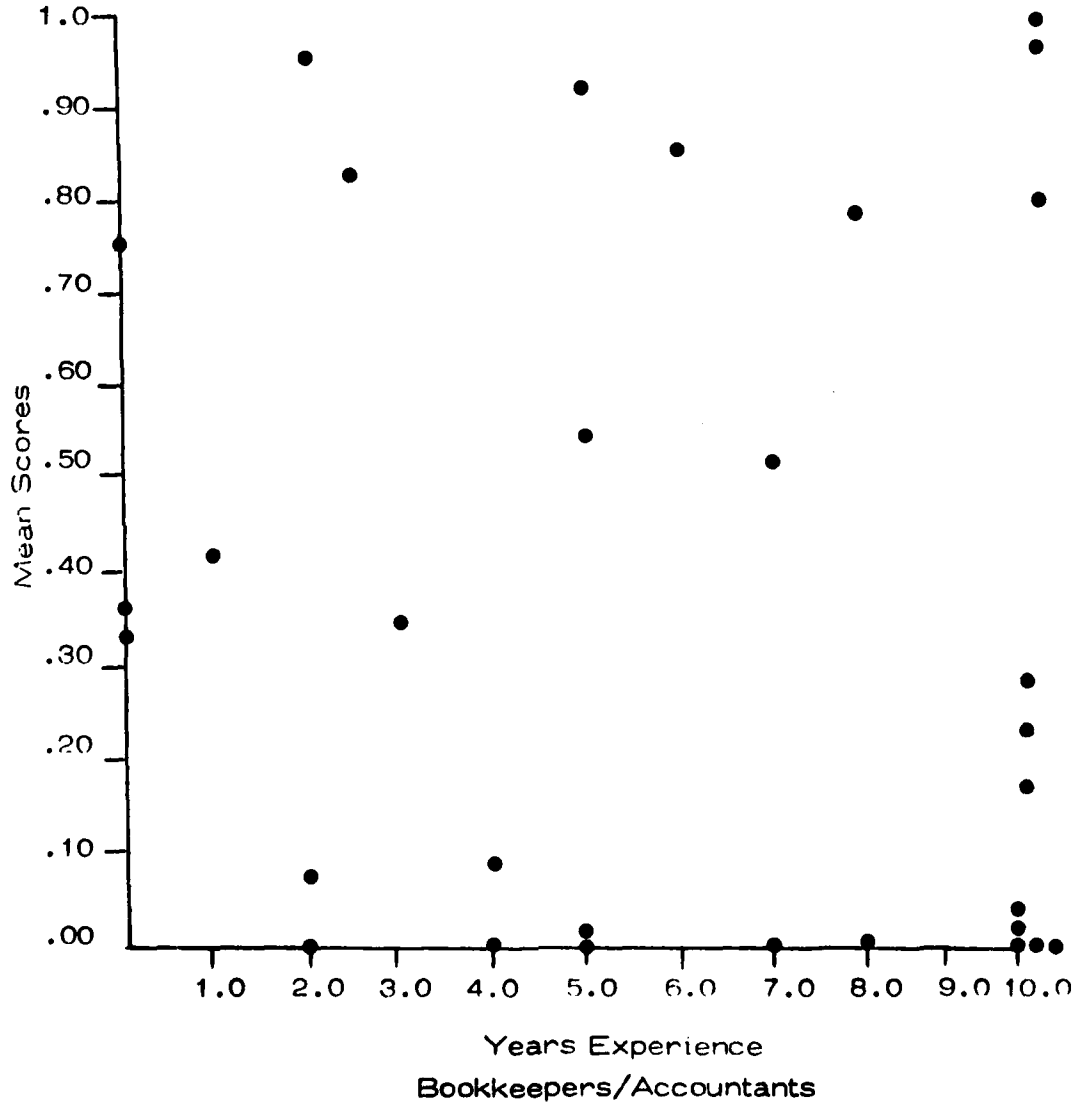


Figure 12. Mean Score vs.Years Experience: Bookkeepers/ Accountants

Table 13

Correlation/Regression Analyses:
Demographic Factors vs. Area-Score

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
Univariate Regressions					
Age	-.225	-.0089	5.1	-3.36	.005
Multi-variate Regression					
Age	-.225	-.0196		-5.89	.0005
Higher Education	.077	.0140	15.2	1.53	.10
Years Experience	.085	.0241		4.62	.0005

Analysis of Variance Table*

Source	Sum Squares	df	Mean Squares	F Ratio	P ≤
Regression	5.28	3	1.762	12.48	.005
Residue	29.49	209	.411		

*Analysis of Variance Table for the Multi-variate Regression

Table 14

Correlation/Regression Analyses:
Demographic Factors vs. Combinational-Strategy Score

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
Univariate Regressions					
Age	-.160	-.0055	2.6	-2.36	.025
Multi-variate Regression					
Age	-.160	-.012		-4.11	.0005
Higher Education	.098	.014	8.6	1.72	.05
Years Experience	.061	.014		3.13	.005

Analysis of Variance Table*

Source	Sum Squares	df	Mean Squares	F Ratio	P≤
Regression	2.253	3	.750	6.52	.005
Residue	24.05	209	.115		

*Analysis of Variance Table for the Multi-variate Regression

significant in univariate regressions, although all three factors had significant, non-zero coefficients in a multivariate regression. Together, they explained about 9% of the strategy-score variance with, again, higher-education and years-of-experience positively correlated with strategy score.

Analysis 7: Experience-Related Demographic Factors and Area-Score

Purpose. To investigate the effect of experience-related demographic factors on area-score.

Method. A correlation analysis was used to reveal the relationship between demographic variables and area-score. Experience-related demographic variables were:

1. The number of programming languages known and used to code at least 1 program.

2. The number of programming languages used to code 11 or more programs.

3. The total number of programs written.

4. The number of programming areas involved in, among the following:

   a. Data entry
   b. Production Control
   c. Operations
   d. Application Programming
   e. System Programming
   f. System Analysis
   g. Data Base Administration
   h. Data Communication
   i. Other

5. The number of operating systems used.

In addition to the correlation analysis, univariate and multivariate analyses were conducted using the experience-related factors as independent variables and area-score as the dependent variable.

Results. Table 15 gives the results of the correlation analysis, in which "number of programming languages" and "number of operating systems" were shown to have correlation coefficients of .435 and .429, respectively. This result suggests that breadth of experience rather than mere length of work experience may be more important to performance prediction.

Since the "number of programming languages used in 11 or more programs" had a considerably lower correlation coefficient than "number of programming languages used" (.435 vs. .207), the amount of experience with a language was apparently less important than knowing multiple languages.

The univariate and multivariate regressions shown in Table 16 revealed that only three factors ("number of languages", "number of programs written" and "number of operating systems") explained a significant amount of score variance. The result noted above, concerning the lack of importance of the amount of experience, was further emphasized by the negative correlation between "number of programs written" and score. The factor "number of programs written" may be closely associated with other factors such as age (shown to be also negatively correlated with area score) or narrowness of experience (individuals who write many programs in one language and use one operating system may not have the broad viewpoint of individuals using multiple languages and systems).

A further result was that the percent variance explained by "number of languages" and "number of operating" systems (18.9%, and 18.4%, respectively) in univariate analyses was almost additive for the variance explained in the multivariate analysis. This suggests that these two factors reflect different capabilities, and that each experience individually contributes to an individual's ability to perform well.

Analysis 8: Strategy-Factors

Purpose. To determine the relationships among experimental factors, including measures of participant-strategy, and to identify the relationship between strategy-factors and area-score.

Table 15

Correlation Between Experience Related Demographic Factors and Area-Score

	Correlation Coefficient
1. Number of Programming Languages Used for 1 or More Programs	.435
2. Number of Programming Languages Used for 11 or More Programs	.207
3. Total Number of Programs	-.281
4. Number of Areas of Experience	.210
5. Number of Operating Systems	.429

Table 16

Correlation/Regression Analyses
Experience Related Demographic Factors
vs. Area-Score

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
Univariate Regressions					
No. of Languages	.435	.1175	18.9	2.55	.01
Total No. of Programs Written	-.281	-.00055	7.9	-1.55	.10
No. of Operating Systems	.429	.0838	18.4	2.51	.01
Multivariate Regression					
No. of Languages	.435	.1045		2.36	.025
Total No. of Programs Written	-.281	-.00066	39.1	-2.15	.025
No. of Operating Systems	.429	.0588		1.85	.05

Analysis of Variance Table*

Source	Sum Squares	df	Mean Squares	F Ratio	P ≤
Regression	1.402	3	.467	5.56	.005
Residue	2.184	26	.083		

*For Multivariate Regression

Method. A correlation analysis was used to determine the relationships among the experimental factors and the measures of participant strategy. Univariate and multivariate regressions were developed to establish the relationships between the strategy-factors and area-score.

Results. Correlations among the experimental independent variables are shown in Table 17. The probabilities P_1 and P_2 were anti-correlated with a correlation coefficient - .933. Since this had been shown previously to be a result of the way these factors were computed, P_1 and P_2 were not used in a regression as independent variables. Also, P_1 was correlated (.412) with combinational-strategy, suggesting that many individuals who used a systematic strategy also generated multiple example-solutions before testing the solutions via feedback-aids.

Table 18 provides the results of the univariate regression analyses, which indicated that P_1 and P_2 were significantly correlated with area-score, explaining 9% and 6.3% of the variance, respectively. Combinational-strategy, which was also significant, explained 58% of the variance, and was thus the best single predictor of area-score. For reference, pretest score, which could be taken as a measure of a participant's ability to develop example-solutions, was found to be significant and predicted 44% of the score variance in a univariate regression.

In a multivariate regression, also shown in Table 18, a combination of P_1 and combinational-strategy resulted in a non-significant coefficient for P_1, and virtually no increase in variance explained over that explained by the combinational-strategy in a univariate regression. Thus, substantially all the variance explained by the P_1 and P_2 factors was also explained by the combinational-strategy factor.

A similar analysis using a multivariate regression, shown in the same table, indicated that combinational-strategy and pretest score provided a significant regression and explained 68% of the variance. Thus, although most of the variance explained by the pretest score was also explained by the combinational strategy factor, approximately 10% additional variance was explained using the two factors together.

Table 17

Correlation Among Independent (Strategy) Variables

		2	3	4	5	6	7
1.	Session	.018	.000	.204	-.206	.156	-.013
2.	Problem-Complexity		.016	.125	-.139	-.124	.012
3.	Feedback-Aid			.022	-.150	-.029	-.013
4.	P_1				-.933	.412	.271
5.	P_2					-.355	-.221
6.	Combinational Strategy-Measure						.534
7.	Pretest Score						

Table 18

Correlation/Regression Analyses:
Strategy Factors vs. Area-Score

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
Univariate Regressions					
P_1	.300	.377	9.0	4.14	.0005
P_2	-.251	-.308	6.3	-3.42	.005
Combinational Strategy	.766	.901	58.6	15.71	.0005
Pre-test Score	.669	.781	44.8	11.87	.0005
Multi-variate Regression					
P_1	.300	-.024	58.7	-.359	NS
Combinational Strategy	.766	.915		14.425	.0005
Multi-variate Regression*					
Combinational Strategy	.766	.676	68.1	11.26	.0005
Pre-test	.669	.425		7.17	.0005

Analysis of Variance Table*

Source	Sum Squares	df	Mean Squares	F Ratio	P ≤
Regression	20.09	2	10.04	184.86	.005
Residue	9.40	173	.04		

*Analysis of Variance Table for the second Multi-Variate Regression

Discussion of Experiments 3 and 4

Feedback-Aids

The effect of the feedback-aids as measured by the area-score was not statistically significant. However, the effect of the aids measured by the relative area-score was found to be significant and important.

Apparently the aids did help at least a portion of the participant population — the less-than-superior performers. Those who would perform well without the aids were not helped by the aids. Also, it is apparent that variations in performance due to the participants' innate abilities may have been greater than variations in performance due to the feedback-aids. This factor, plus the observation that superior performers may not need or use the aids, may account for the insignificant effect of the aids on area-score and the significant effect on relative area-score.

Feedback-Aid #3 appears to have affected performance to a greater degree than did Aid #2. Aid #3 included the SSL and provided recommended next-logical example-solutions based on patterns in the example-solutions input previously. On the other hand, Aid #2 included the SSL and an ordered list of example-solutions previously entered. With Aid #2 the participant had to examine the pattern of previous inputs and develop any missing example-solutions. With Aid #3 the participant was presented with recommendations. Thus, if the example-solutions previously entered were not in error, it was reasonable to expect that Aid #3 might support performance that was superior to that supported by Aid #2.

Yet Aid #3 could give false recommendations without indicating the basis for the false recommendations. If, for instance, Aid #3 was used early in the problem, when only a few example-solutions had been input by the participant, there were numerous possible completion-patterns many of which were not correct. As a result, each of the recommended example-solutions should have been carefully tested before it was accepted. Also, if the participant had input an example-solution that contained an error, Aid #3 recommended for consideration solutions that actually completed the error-induced patterns. This information could be helpful or harmful depending on how it was used. If the

recommended solutions were used without careful evaluation, this type of feedback would have been harmful. If however, the display of patterns built on an error increased the likelihood of the error's detection — by displaying its impact — then aids providing recommended-solutions would have been helpful in detecting input-errors.

Demographic Factors

The lack of a strong relationship between years-of-higher-education or years of experience and performance may be a surprise to educators and directors of personnel departments. This result was also reported by Connelly et. al 1981 and by Sheppard, Kruesi, and Curtis 1980.

Additional results suggested that the "number of programming languages (used on 1 or more programs)" and "number of operating systems used" were factors that should be used for evaluating computer users/programmers capability - in place of years-higher-education and years-of-experience.

But there was additional information in the results. The fact that "number of programs used for 11 or more programs" did not result in a significant univariate regression and had a lower correlation value with performance than did "number of languages used for at least 1 program" suggests that depth of language experience was not as important as breadth.

Further, the question arises: were the language- and operating-system factors stated above merely indicators of superior performers (i.e., superior performers may tend to learn many languages and use many operating systems) or did, instead, the knowledge of multiple languages and experience with multiple operating systems have value in themselves. If the former hypothesis is true, then one would expect that the number of languages known and the number of operating systems used would reflect the same factor: namely, individual capability. Further, very capable individuals might be expected to write many programs in each language and work in a number of programming areas. But, in fact, these other variables did not result in significant univariate regressions or were negatively correlated with performance (e.g., "Number of programs written" was negatively correlated with performance and "Number of Programming

areas" was not significant). Thus, there was no strong support for the first hypothesis.

Additional and positive evidence supporting the second hypothesis was that the variances explained by the "number of languages" and "number of systems" factors in univariate regressions (each of which explained approximately 18% of the performance variance) combined almost additively in the multivariate regression using both factors and "number of programs written" for the independent variables. This result implies that the two factors — "number of languages" and "number of operating systems" — were substantially different factors and supports the hypothesis that knowledge or experience factors are of value in themselves and were not merely indicators of superior performers.

With respect to the implications of the second hypothesis, a common factor, "knowledge of, or the ability to generate, alternative approaches to a problem" (referred to subsequently as "alternative approaches"), would be expected to increase as additional programming languages are learned and as additional operating systems are used. Further, the repeated use of a language already known would not be expected to increase knowledge of alternative approaches greatly because it is repetitious and not conducive to formulating new viewpoints. Under this argument, the "number of languages used for 11 or more programs" would not be expected to be an important variable. A similar argument applies to the "number of programs written" and to "the number of application-areas".

The failure of the factor "number of years-of-experience" to explain a large amount of performance variance may simply reflect the repetitious nature of experience, unconducive to the creation of alternative viewpoints. The failure of "years-of-advanced-education", however, to explain a considerable amount of performance variance is a curious result, because one might expect that advanced education would help to create alternative viewpoints or at least to be helpful in some way in problem solving. It may be that education in software science and accounting/bookkeeping tends to involve rote learning rather than more creative problem-solving methods which develop the ability to create alternative viewpoints.

The conjecture that the ability to develop alternative approaches was an underlying performance-factor appears to hold up against the tests of the independent variables described above. Of course, experiments need to be designed to test the alternative-approach hypothesis; but, it is an attractive hypothesis, since it presumably involves a "trainable" factor. Further, there may exist other experience-related, independent variables that can enhance the ability to develop alternative approaches, that are different from "number of languages" and "number of operating systems", and that can further improve our ability to predict performance. Ultimately, these may lead to improved problem-solving performance in command and control, and programming tasks.

Strategy Factors

Combinational-strategy was found to be the single best predictor of performance. This result is the same as that found in Experiments1 and 2, where approximately the same percent of variance was explained (63%in Experiments1 and 2 and 58% in Experiments 3 and 4). The success of the combinational-strategy measure shows the power of moment-to-moment measures, i.e., their ability to indicate the value of actions currently underway. When validated with summary performance-measures, the moment-to-moment measures become operational-measures in themselves. A moment-to-moment measure thus provides greatly improved sensitivity over summary measures for the evaluation of each user-input.

EXPERIMENT 5

Purpose

The purpose of Experiment 5 was to investigate the ability of experienced programmers to detect and then to revise initially incorrect example-solutions to problems similar to those used in the previous experiments.

Method

Participants

Participants were experienced programmers obtained by the same means as for Experiments #1 and #3.

Procedure

The procedure used in this experiment was the same as that in Experiments #1 through #4.

Experiment #5 Design

The experiment design was the same as that for Experiments #3 and #4, except that:

1. Only one participant population (programmers) was used.

2. Factor A, which, in Experiments #1 through #4, was the problem-complexity as measured by Halstead's E Metric, now became the number of ship-combinations (out of a total of 14 possible) that were correct at the beginning of each problem.

Experiment Task

Each experiment task consisted of reading the specifications for a Naval task force, and then of examining and correcting, as necessary, a set of example-solutions (i.e., ship combinations) for that task force. Each task-force specification required 14 distinct example-solutions (i.e., 14 distinct sets of ship-combinations within a specified range of transiting and stationing times). Thus, the participant's task

was to review an existing set of example-solutions, delete incorrect example-solutions, and enter any new example-solutions necessary to completely specify all the required ship-combinations within the required range of transiting and stationing times.

This task was similar to that used in Experiments #1 through #4, except that, in those earlier experiments, the participant was not presented with an initial set of example-solutions and, thus, the participant himself had to enter all the example-solutions. In experiment #5, participants were presented with a set of 14 example-solutions entered into the computer, among which, however, were several incorrect solutions. The three problem-complexity levels of Experiment #5 corresponded therefore to 6, 9, and 14 initially incorrect example-solutions.

A pretest problem was given to each participant prior to the three experiment problems. The pretest problem was the same as that given to participants in the previous experiments, i.e., it did not present an initial set of example-solutions.

Performance Measurement

Several measures were used to evaluate performance in Experiment #5. One measure (M_1) was the probability of retaining an example-solution that was initially correct. Another measure (M_2) was the probability of cancelling an example-solution that was initially incorrect. A third measure (M_3) was the ratio of the number of correct example-solutions to the total number of possible correct example-solutions (viz., 14). The final measure (M_4) was the probability that an example-solution was entered in error.

Measures M_1 and M_2 were unique to the problem of revising example-solutions; they had no counterpart in the problems in which the participants entered all the solutions. Measure M_1 had one disadvantage which limited its use. M_1 was calculated by dividing the number of example-solutions that were initially correct and were not modified by the participant by the number of example-solutions that were initially correct. In the most complex problem, however, there were no example-solutions that were initially correct. Since it is not possible to divide by zero, that problem was not scored with measure M_1.

Data Analysis

Analysis 1: Average Cell-Scores for M_1

Purpose. To determine the grand mean and average cell probabilities for retaining example-solutions that were initially correct.

Results. Table 19 gives the grand-mean and average-cell probabilities for M_1. The grand mean probability over the six cells was .934.[1] There does not appear to have been a consistent trend-effect for the A or B factors.

Analysis 2: Average Cell-Scores and ANOVA for M_2

Purpose. To determine whether there were significant effects due to problem-complexity and the feedback-aids on M_2 (the probability of cancelling an initially incorrect example-solution).

Method. Average cell probabilities and the grand-mean value for M_2 were calculated. An ANOVA for measure M_2 was also calculated.

Results. Results are given Table 20. The grand-mean value for M_2 was .650, which was substantially less than the grand-mean for M_1 (.934). This implies that the probability of recognizing that a correct entry was correct and of not changing that entry was greater than the probability of recognizing and cancelling an incorrect entry.

The analysis of variance table shows that the effect of problem complexity was statistically significant, but that the effect of the feedback-aids on M_2 was not significant. Further, the trend of M_2 was to increase with an increasing number of initially incorrect example-solutions. Apparently, at least in the experiment environment, an increase in initial errors led to an increased probability of detection and correction for a single error.

Table 19

Average Cell-Scores M_1

Average Cell-Scores

	A_1	A_2	A_3*
B_1	.872	.960	—
B_2	.936	.960	—
B_3	.936	.940	—

A_1, A_2, and A_3 are the problem-complexity levels in increasing order.

B_1, B_2, and B_3 are the Feedback-Aids #1, #2, #3, respectively.

*M_1 is mathematically undefined for level A_3.

Grand Mean = .934

Table 20

Analysis of M_2

Average Cell-Scores

	A_1	A_2	A_3
B_1	.500	.688	.764
B_2	.633	.624	.707
B_3	.500	.777	.658

A_1, A_2, and A_3 are the problem complexity levels in increasing order.

B_1, B_2, and B_3 are the Feedback-Aids #1, #2, #3, respectively.

Grand Mean = .650

Analysis of Variance Table

Source	Sum Square	df	MS	F
Between Subjects	7.1735	29		
Groups	.2718	2	.1359	.532 NS
Subject W/I Groups	6.9017	27	.2556	
Within Subjects	5.2657	60		
A	.5087	2	.2538	2.901 *
B	.0013	2	.0006	.008 NS
(AB)'	.0196	2	.0098	.112 NS
Error (Within)	4.7358	54	.0877	

* $P \leq .10$

Analysis 3: Average Cell-Scores and ANOVA for M_3

Purpose. To determine whether there were significant effects due to problem-complexity and the Feedback-Aids on M_3 (the ratio of the number of correct example-solutions to the total number of possible correct example-solutions).

Method. Average cell probabilities and the grand-mean value for M_3 were calculated. An ANOVA for M_3 was also calculated.

Results. Table 21 provides the results. The grand-mean value was .793, which, as expected, was intermediate to the grand-mean values for M_1 and M_2.

The ANOVA revealed that the effect of problem-complexity was statistically significant, but that the effect of the Feedback-Aids was not significant.

Analysis 4: Average Cell-Scores and ANOVA for M_4

Purpose. To determine whether there were significant effects due to problem-complexity and the Feedback-Aids on M_4 (the probability that an erroneous example-solution was entered).

Method. Average cell probabilities and the grand-mean value for M_4 were calculated. An ANOVA for M_4 was also calculated.

Results. Results are shown in Table 22. Neither problem-complexity nor the Feedback-Aids was found to have a significant effect on M_4.

Table 21

Analysis of M_3

Average Cell-Scores

	A_1	A_2	A_3
B_1	.857	.799	.764
B_2	.878	.735	.707
B_3	.871	.850	.678

A_1, A_2, and A_3 are the problem-complexity levels in increasing order.

B_1, B_2, and B_3 are the Feedback-Aids #1, #2, #3, respectively.

Grand Mean .793

Analysis of Variance Table

Source	Sum Square	df	MS	F
Between Subjects	3.0277	29		
Groups	.0815	2	.0407	.374 NS
Subjects W/I Groups	2.9461	27	.1091	
Within Subjects	2.3245	60		
A	.3488	2	.1744	4.826 *
B	.0184	2	.0092	.256 NS
(AB)'	.0058	2	.0024	.082 NS
Error (Within)	1.9513	54	.0361	

* $P \leq .05$

Table 22

Analysis of M_4

Average Cell-Scores

	A_1	A_2	A_3
B_1	.123	.102	.040
B_2	.032	.160	.196
B_3	.062	.011	.016

A_1, A_2, and A_3 are the problem-complexity levels in increasing order.

B_1, B_2, and B_3 are the Feedback-Aids #1, #2, #3 respectively.

Grand Mean = .0989

Analysis of Variance Table

Source	Sum Square	df	MS	F	
Between Subjects	2.864	29			
Groups	.237	2	.1189	1.223	
Subject W/I Groups	2.626	27	.0972		
Within Subjects	1.189	60			
A	.0562	2	.0281	1.405	NS
B	.0444	2	.0222	1.111	NS
(AB)'	.0077	2	.0038	.193	NS
Error (Within)	1.0809	54	.0200		

Analysis 5: Relationships between Feedback-Aids, Session-Number, Number of Ship-Combinations Initially Correct and Measures M_1, M_2, M_3 and M_4

Purpose. To determine whether significant relationships existed between various combinations of the Feedback-Aids, session-number, and number of ship-combinations initially correct and the "M" measures.

Method. Multivariate regression analyses were run using combinations of the Feedback-Aids, session-number, and problem-complexity (number of ship-combinations initially correct) as the independent variables and the "M" measures as the dependent variables. Three types of regression analysis were performed using the coding system for the Feedback-Aids described for Experiments 3 and 4. In one analysis, termed "Analysis A", Feedback-Aid codes 1, 2, and 3 were assigned to Aids #1, #2, and #3, respectively, and together were used as one independent variable. In another type of analysis, termed "Analysis B", Feedback-Aid codes 1 and 2 were assigned to Aids #1 and #2, respectively, and the data for Aid #3 was removed from the file. Finally, in the third analysis, termed "Analysis C", codes 1 and 2 were assigned to Aids #1 and #3, respectively, and were then used as the independent variable with the data for Aid #2 removed from the file.

Results. Regressions using M_1 for the dependent variable revealed that none of the three independent variables (Feedback-Aids, session-number, and problem-complexity) had significant regression coefficients. Since M_1 was not defined for problem level A_3, however, only partial data were used in the regression analyses.

Results of the multivariate regression analyses using M_2 as the dependent variable are given in Table 23. In each analysis, although Feedback-Aids were found to be not significant, problem-complexity was significant. Session-number was also significant in two of the three analyses. Apparently, since session-number was positively correlated with M_2, there was continued learning and resultant improved performance on each successive experiment.

Table 23

Correlation/Regression Analyses:
M_2 vs. Experimental Factors

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
		Analysis A			
Feedback-Aids 1,2,3	-.006	-.0014		-.030	NS
Session	.194	.0936	7.6	1.98	.05
No. of Ship-Combinations initially correct	-.183	-.0161		-1.88	.05
		Analysis B			
Feedback-Aids 1,2	.005	.0015		.016	NS
Session	.142	.0704	5.2	1.17	NS
Combinations initially correct	-.168	-.0146		-1.36	.10
		Analysis C			
Feedback-Aids 1,3	-.007	-.0018		.02	NS
Session	.265	.1274	13.9	2.33	.025
No. of Ship-Combinations initially correct	-.251	-.0222		-2.12	.025

Table 24 gives the results for the multivariate regression analyses in which M_3 was the dependent variable. Again, the Feedback-Aids were found to be not significant, while session-number and problem-complexity were both found to be significant in each of the three analyses.

Since session-number was significant and positively correlated with M_3 (and also with M_2), performance apparently continued to improve with practice. This is in contrast to the results of Experiments 1-4, where session was not found to be a significant factor.

Problem-complexity was found to be positively correlated with M_3. Thus, as the number of initially correct solutions increased, there was an increase in the ratio of the number of correct solutions to total number of solutions. This result was as expected.

Finally, regressions using M_4 (the probability that an example-solution entry was in error) revealed that there were no statistically significant relationships between session-number, feedback-aids, problem-complexity, and M_4. The correctness of an example-solution entry was not affected by problem parameters or session-number.

Discussion: Experiment 5

It is apparent that differences among the Feedback-Aids, which were the same aids as those used in Experiments 3 and 4, did not benefit participants engaged in revising example-solutions. Even analyses comparing the performance of pairs of aids, which were similar to the analyses of the data from Experiments 3 and 4, did not reveal any significant results for the Feedback-Aids. Further, analyses using relative measures, which tended to remove the effect of participant skill, did not result in statistically significant results.

Apparently, then, differences among the aids of the type used here, which may have helped in solving problems initially, did not help in revising problem solutions. The reason may be as follows. All the aids included the ship selection logic (SSL) feature, which was automatically generated from the existing set of example- solutions. The example-solutions presented in an ordered way, and recommendations for next-logical example-solutions based on incomplete combinational

Table 24

Correlation/Regression Analyses:
M_3 vs. Experimental Factors

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
		Analysis A			
Feedback-Aids 1,2,3	-.012	-.0047		-.16	NS
Session	.287	.0811	14.6	2.72	.005
No. of Ship-Combinations initially correct	.269	.0137		2.53	.01
		Analysis B			
Feedback-Aids 1,2	-.069	-.0312		-.51	NS
Session	.264	.0738	13.0	1.98	.05
No. of Ship-Combinations initially correct	.255	.0126		1.90	.05
		Analysis C			
Feedback-Aids 1,3	-.015	-.0090		-.156	NS
Session	.370	.1060	18.2	2.97	.005
No. of Ship-Combinations initially correct	.229	.0115		1.76	.05

patterns detected in the existing example-solutions were the features added by Aids #2 and #3, respectively. These new features were designed to help the participants detect omissions in combinational patterns. However, when revising existing example-solutions some of which were not correct initially, the omissions noted in the combinational patterns would have been based, at first, on the existing incorrect example-solutions, and thus the suggested patterns would have been incorrect. If participants had first cancelled all incorrect example-solutions and then had entered the correct solutions, the aids might have been more helpful. The conclusion, then, is that the strategies used successfully to revise existing solutions are at present unknown, and that the aids designed to help in the development of new solutions did not appear to be helpful in the revision of solutions.

Analysis of the grand-mean probabilities of success with the four measures revealed a probability of .934 for retaining an initially correct solution and a probability of .902, i.e. 1. -.098 (see Analysis 4), for making an entry correctly. Since there was a grand-mean probability of only .650 for detecting and cancelling an erroneous example-solution, there was an obvious performance decrement in the detection and correction of erroneous example-solutions. Useful aids might therefore be directed toward the detection and correction of existing errors.

But there appears to have been a still more serious problem in the detection of errors. Problem-complexity measured by the number of ship-combinations initially correct was negatively correlated with M_2. Thus, as the number of initially correct example-solutions was increased, there was a decrease in the probability of correcting incorrect example-solutions. Stated differently, as the number of incorrect solutions was decreased there was a decrease in the probability of detecting and correcting those incorrect example-solutions. The fewer the errors, the lower the probability of detecting a given error. This result suggests that there is a base-line probability (based on the frequency of errors recently detected) for incorrect example-solutions which affects the probability of judging that any solution is in fact incorrect. Thus, the perceived correctness of a solution may be determined as function of:

1. the properties of that solution, and

2. the frequency of erroneous solutions previously discovered.

This result, which is consistant with predictions of signal detection theory (which states that the probability of an event is a function of some base-line probability in addition to specific measurements on the signal itself), would seem to predict a decreasing probability of detecting and correcting errors with decreasing error-rates. Also, it suggests that the probability of detecting the last few errors may be so small that it is not likely that those few remaining errors will be found. But this result also suggests a possible solution: seeding errors to increase the base-line-error probability and, thus, to increase the probability of detecting unknown errors.

EXPERIMENT 6

Purpose

The purpose of Experiment 6 was to investigate the capabilities of programmers experienced in Fortran IV to write Fortran IV programs to solve the same problems used in the Experiments 1 through 4. The intent was to present to the programmer-participants a task that was analogous to the task presented to participants in the earlier experiments, where participants had to develop example-solutions to solve the experiment problems. Having Experiment 6 participants develop solutions to these same problems, but in Fortran IV code, permitted a comparison of the accuracy and completeness of solutions specified by example-solutions with those specified by the Fortran IV code.

Method

Participants

Participants were experienced Fortran IV programmers obtained by the same means as the programmers for the earlier experiments, i.e. via temporary personnel agencies and newspaper advertisments. Participant requirements were the same as for the other experiments, except that only experienced Fortran IV programmers were accepted.

Task

The objective of Experiment 6 was to determine the ability of programmers to produce Fortran IV code for the problems used in Experiments 1-4. While the complexity of those problems was fixed by the problems themselves, the machine processing of data in Experiments 1 & 2 affected the number of example-solutions the participants had to develop and input. At one level, where little automatic processing was provided, the participant had to develop a large number of example-solutions. At another processing level, considerable automatic processing was provided and, thus, fewer example-solutions were required to specify the desired logic. In order to provide a task in Experiment 6 that would mimic this variation in processor complexity, the specification requirements for the stationing- and transiting-times had to be varied. Table 25 shows the participant-input required by this complexity-factor in Experiment 6 along with the inputs required by the analogous

Table 25

Similarity of Mission-Types (Experiment 6) and Processor-Complexity Levels (Experiments 1 & 2)

Experiment #6		Experiment #1	
Mission Type	Specification Requirement	Automatic Processing Level	Specification Requirement
1	The MIN and MAX transiting-and stationing-times are constants, i.e. are independent of ship type or ship-combination makeup.	1	The MIN and MAX transiting-and stationing-times apply to all ships regardless of type or combination.
2	The MIN and MAX transiting-and stationing-times are a function of each distinct ship-combination.	2	The MIN and MAX transiting-and stationing-times apply to all ships in each particular combination.
3	The MIN and MAX transiting-and stationing-times are a function of of each ship type.	3	The MIN and MAX transiting-and stationing-times apply to each ship type in each combination.

factor in Experiment 1. Tables 26, 27, and 28 show the Experiment 6 specifications, referred to as Mission Types 1, 2, and 3, respectively, for the stationing- and transiting-times as given to the participants.

As shown in Table 25, the task transiting- and stationing-time requirements in Experiment 6 are analogous to the processor complexity levels of Experiments 1 & 2. This should not be taken to imply that the associated tasks were identical, however. In Experiments 1 & 2, participants were asked to develop and to input enough specific example-solutions, i.e. discrete sets of ship-combinations, to specify, with the aid of the processor, the ship-selection logic (SSL). In contrast, Experiment 6 participants were asked to develop syntactic specifications for the ship-selection logic using Fortran IV code.

Table 26

Mission Type 1

All ships must have a transiting-time of 5 days or less,

AND

All ships must have a stationing-time of 10 days or more.

Design

The design of Experiment 6 was a 2 x 2 factor Latin Square design with three levels per factor. This design has been described in previous material on Experiments 3 and 4. Factor A was the problem-complexity. Factor B (Mission-Type) comprised the MIN and MAX transiting- and stationing-time requirements.

Procedure

Each participant was scheduled for either an 8:00 a.m. or 1:00 p.m. session. Upon arrival, the participant was asked to fill out a biographical-data form and an experiment agreement. On completion, the biographical data were reviewed by the experimenter to insure compliance with the preestablished experiment entrance criteria. The participant, if accepted, then viewed a video tape which provided the experiment instructions. The participant was given a copy of the instructions to follow along and to mark-up as he/she chose.

Table 27

Mission-Type 2

The stationing time is a function of the makeup of the ship-combinations, as follows:

If there is 1 or more nuclear ship (a CVAN, CGN, or SSN) in a particular ship-combination, then the stationing-time for all ships in that combination must be 30 or more. If there are no nuclear ships in a particular combination, then the stationing-time for all ships in that combination must be 10 days or greater.

The transiting time specification is 5 days or less for all ships.

Following the presentation of instructions, the participant entered the experiment room to work on the four problems. The first problem was a pre-test problem, followed by the three experiment problems. One hour, maximum, was allowed for each problem.

The participant entered Fortran IV code via a keyboard and observed the code on a terminal video-display. The computer system was placed in a screen-edit mode by the experimenter prior to each experiment. Thus, the participant could:

1. Enter code by pressing the alpha-numeric keys.
2. Correct an error by placing a curser under the target-character and pressing the ERASE key.

Table 28

Mission-Type 3

	Transiting-Time Specification	Stationing-Time Specification
CVA	10 Days or Less	10 Days or More
CVAN	5 Days or Less	30 Days or More
CA	7 Days or Less	10 Days or More
CG	7 Days or Less	10 Days or More
CGN	10 Days or Less	30 Days or More
DD	5 Days or Less	20 Days or More
SS	6 Days or Less	20 Days or More
SSN	10 Days or Less	30 Days or More
AO	20 Days or Less	20 Days or More

(Four cursor controls were available, each of which would move the cursor one character when pressed. The cursor controls were: up, down, left, right).

3. Start a new line by pressing the RETURN key.

4. View (and, if necessary, correct) previously entered code which had been scrolled off-screen by moving the cursor up or down.

The participant could enter and visually check the source-code, but could not compile it.

Data Processing

The code entered by each participant for each problem was maintained as a file. There were 30 (participants) x 4 (problems each) = 120 files. Each file was subsequently modified to be a subroutine so it could be tested by a main test-program. The modification consisted of entering on a new first line the subroutine name "SSL" and the names of the variables passed from the main test-program to the subroutine and, on a new last line, a "RETURN" command. (Note: each participant was provided with a printed glossary of variable names).

A permanent set of files, as entered by the participants, was stored on tape in order to maintain a record of the "raw" data of the experiment. A duplicate set of files was then made; each file was modified to form a subroutine; each subroutine was compiled; and a compiler-listing was produced, annotated with error statements. Based on the compiler-reported errors, the clearly unintentional errors were determined and corrected, providing a compilable object-module. A record of all corrections was maintained and used in an error-analysis, to be described below.

Some of the error-corrections did not affect the logic of the subroutine. These corrections included: syntax (e.g. missing commas), missing parenthesis in "IF" statements, statements starting in Columns 1-6, and continuation marks not in Column 6. Other corrections which did affect the subroutine logic were: misspelled words, array indexing-errors, and defective dotted keywords (used in logical statements, e.g. "AND."). The correction policy was

to make changes only when the compiler identified an error for which the participant's intent (and therefore the necessary correction) was obvious. For instance, a defective dotted keyword might be bound, such as "AND.", which was then corrected to ".AND.". Misspelled words often were easily corrected with confidence as to intent, e.g. "ISHIP" was sometimes spelled "ISHIPS". IF statements placed line-after-line, i.e. in parallel construction, permitted correction of some missing parentheses.

In other cases, corrections could not be made with confidence as to the intended logic. For instance, in the case of missing statements, missing variables, or parentheses - errors in complex statements that did not have other similar statements for a guide, the corrections were not obvious and were entered in a standard way, as follows. Missing statement errors resulting from a statement such as "GO TO N" when there was no statement labelled N were resolved for compilation by entering an "N Continue" statement after the calling statement. For a missing variable, a dummy variable was entered. Parentheses-corrections were supplied at the beginning or ending of "IF" statements. The only changes made in each file were those required to get an object-module free of compiler-detected errors.

The logic of each subroutine was tested by linking it to a main test-program,which would systematically develop both correct and incorrect test ship-combinations, transfer the test ship-combinations to the subroutine, and record their acceptance or rejection by the subroutine.

Performance Measurement

Three performance measures were used to evaluate each subroutine written by the participant. One measure (P_C) was the probability that the subroutine would accept a ship-combination given that it was in fact a correct combination (with probability $\bar{P}_C$) according to the problem requirements. The second measure (P_{IC}) was the probability that the subroutine would reject a ship-combination given that it was in fact an incorrect combination (with probability $\bar{P}_{IC}$) according to the problem requirements. The third measure (P_T) was the probability that the subroutine would accept or reject a ship-combination.

Hence:

$$P_C (\bar{P}_C) + P_{IC} (\bar{P}_{IC}) = P_T \qquad (6)$$

Test ship-combinations were generated by varying the number of ships of each ship type and the transiting- and stationing-times for each ship type, starting with each known correct ship-combination for the problem whose participant solutions were to be tested. Thus, if a known correct ship-combination requiring three ships was:

1 non-nuclear carrier (CVA) — 5 days transiting, 10 days stationing

AND

2 non-nuclear submarines (SS) — 7 days transiting, 15 days stationing

then that combination was used as part of the test. Another combination, a variation of the above, that was also used and was also correct, because shorter transiting times were always acceptable, was:

1 non-nuclear carrier (CVA) — 4 days transiting, 10 days stationing

AND

2 non-nuclear submarines (SS) — 7 days transiting, 15 days stationing

An example of an incorrect ship-combination -- incorrect because the number of submarines is incorrect -- that would have been used in the test is:

1 non-nuclear carrier (CVA) — 5 days transiting, 10 days stationing

AND

1 non-nuclear submarine (SS) — 7 days transiting, 15 days stationing

Table 29 gives the number of correct and incorrect ship-combinations used to test the experiment data for each experiment problem. Each correct ship-combination was varied by adding and subtracting one ship of each type one-at-a-time, and by adding and subtracting one day from transiting- and stationing-times for each ship type one-at-a-time. Using this procedure, approximately 30% of all test ship-combinations formed were correct and approximately 70% were incorrect. That is $\bar{P}_C \cong .30$ and $\bar{P}_{IC} \cong .70$.

If a subroutine logic were to reject all ship-combinations submitted to it, independent of whether the combinations were correct or not, the probability (P_T) that the subroutine would correctly classify a ship-combination selected at random from the test pool was approximately .70. Conversly, if the subroutine accepted all ship-combinations, the probability of correctly classifying a ship combination selected at random from the test pool was approximately .30. When interpreting the experiment data, it was important to realize that, while the value of P_T had a range of anywhere from 0.0 to 1.0, values close to .70 could result from subroutine rejection of all or many ship-combinations independent of whether or not they were correct.

Data Analysis

Analysis 1: Main and Interactive Effects

Purpose. To determine the significance of the main and interactive effects of the experiment variables on P_C, P_{IC}, and P_T.

Method. Analysis-of-variance and Student-Newman-Keuls test were used to analyze the data.

Results. Tables 30, 31, and 32 are analyses of variance for P_C, P_{IC}, and P_T, respectively. Table 30 shows that problem-complexity did not affect performance in a significant way, but that mission-type did have a significant effect on the probability of accepting a correct ship-combination (P_C). In contrast, as shown in Table 31, neither problem-complexity nor mission-type significantly affected P_{IC}, the probability of rejecting an incorrect ship-combination. Also, the level of overall performance was different: incorrect combinations were rejected in 82% of the test problems, but only 51% of the correct combinations were accepted. Finally, Table 32 shows that the results for P_T were similar to the results for P_C,

Table 29

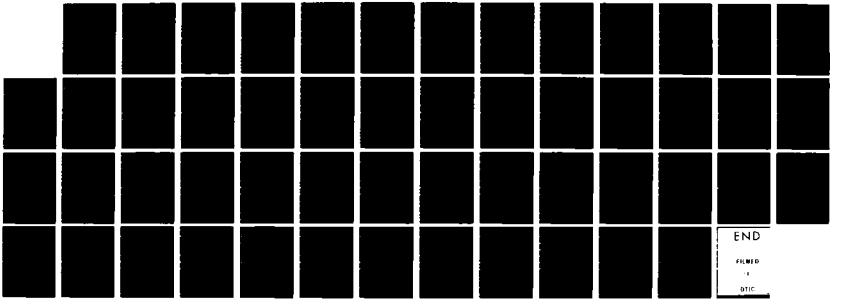
Distribution of Correct and Incorrect Test Ship-Combinations

Problem	Number of Correct Combinations Used	Number of Incorrect Ship Combinations Used	Total Number of Ship Combinations Used	Ratio Correct	Ratio Incorrect
31	138	312	450	.306	.694
52	111	234	345	.322	.678
93*	17	48	65	.262	.738
15	58	132	190	.305	.695

* Pre-test Problem

AD-A124 051 A COMPARISON OF THE ACCURACY AND COMPLETENESS OF PROBLEM SOLUTIONS PRODUC.. (U) PERFORMANCE MEASUREMENT ASSOCIATES INC VIENNA VA E M CONNELLY DEC 82 2/2

UNCLASSIFIED PMA-82-362 N00014-79-C-0739 E/G 9/2 NL



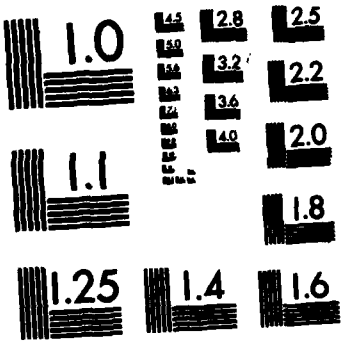


Table 30

Analysis of Variance: P_C

Average Cell Scores

	A_1	A_2	A_3
B_1	.754	.763	.665
B_2	.489	.741	.524
B_3	.141	.229	.296

A_1, A_2, A_3 are the problem-complexity levels in increasing order.

B_1, B_2, B_3 are Mission-Types 1, 2, 3, respectively.

Grand Mean = .510

Analysis of Variance Table

Source	Sum Squares	df	MS	F
Between Subjects	4.5154	29		
Groups	.0053	2	.0026	.016 NS
Subjects W/I Groups	4.5101	27	.1670	
Within Subjects	13.6570	60		
A	.2243	2	.1121	.667 NS
B	4.0309	2	2.0150	11.98 ***
(AB)'	.3194	2	.1597	.95 NS
Error (Within)	9.0824	54	.1681	

***Significant at $P \leq .001$

Table 31

Analysis of Variance: P_{IC}

Average Cell Scores

	A_1	A_2	A_3
B_1	.748	.808	.705
B_2	.888	.831	.863
B_3	.908	.828	.825

A_1, A_2, A_3 are the problem-complexity levels in increasing order.

B_1, B_2, B_3 are Mission-Types 1, 2, 3, respectively.

Grand Mean = .823

Analysis of Variance Table

Source	Sum Square	df	MS	F
Between Subjects	2.966	29		
Groups	.0139	2	.0069	.064 NS
Subject W/I Groups	2.9521	27	.1093	
Within Subjects	3.2799	60		
A	.0375	2	.0878	.342 NS
B	.2162	2	.1081	1.970 NS
(AB)'	.0623	2	.0311	.568 NS
Error (Within)	2.9637	54	.0548	

Table 32

Analysis of Variance: P_T

Average Cell Scores

	A_1	A_2	A_3
B_1	.747	.793	.693
B_2	.765	.802	.760
B_3	.673	.635	.634

A_1, A_2, A_3 are the problem-complexity levels in increasing order.
B_1, B_2, B_3 are Mission-Types 1, 2, 3, respectively.
Grand Mean = .726

Analysis of Variance Table

Source	Sum Square	df	MS	F
Between Subjects	1.134	29		
Groups	.0112	2	.0056	.135 NS
Subject W/I Groups	1.123	27	.0416	
Within Subjects	.9402	60		
A	.0222	2	.0111	.911 NS
B	.2250	2	.1125	9.237***
(AB)'	.0352	2	.0176	1.447 NS
Error (Within)	.6577	54	.0121	

***Significant at $P \leq .001$

namely, that mission-type was the only major factor that affected performance. All three tables indicate that the effect of differences among the participant groups was not significant and, further, that problem-complexity and mission-type interactions were also not significant.

Tables 33 and 34 provide the results of Student-Newman-Keuls tests of mean differences for P_C and P_T, respectively. Data for P_{IC} are not shown because no significant differences were found. For P_C, the SNK test reveals that B_1 - performance was significantly different from that for B_3. For P_T, the order of performance was B_2, B_1, B_3 with B_2 the highest; B_2 - performance was shown to be different from that for B_3.

Analysis 2: Participant Strategy

Purpose. To determine the frequency of use of, and the relationships among, the participant strategy-factors. To determine their effect on performance, i.e. on the completeness and accuracy of the Fortran IV subroutines written by each participant.

Method. In solving the experiment problems by writing Fortran IV subroutines, participants tended to use specific strategies. Although the appropriateness of a particular strategy depended on the problem-complexity and the mission-type, a participant's strategy-choices could be classified independent of whether or not they were appropriate. Analysis of a subroutine disclosed the strategies embedded in it, which were then coded according to the scheme presented below. Strategy factors can be broken down into two broad categories (See Fig. 13): the use of positive-logic or the use of negative-logic. When considering a plan for accepting or rejecting ship-combinations, a participant might use:

1. Positive logic, where statements accepting correct combinations are written and all other combinations are rejected, or

2. Negative logic, where statements rejecting incorrect combinations are written and all other combinations are accepted as being correct.

Table 33

Student-Newman-Keuls Test for P_C

		B_1	B_2	B_3
Means		.727	.584	.222
B_1	.727	-		
B_2	.584	.14 NS	-	
B_3	.222	.502**	.362 NS	-

** Significant at $P \leq .05$.

Table 34

Student-Newman-Keuls Test for P_T

		B_2	B_1	B_3
Means		.776	.744	.657
B_2	.776	-		
B_1	.744	.032NS	-	
B_3	.657	.119**	.087 NS	-

** Significant at $P \leq .05$.

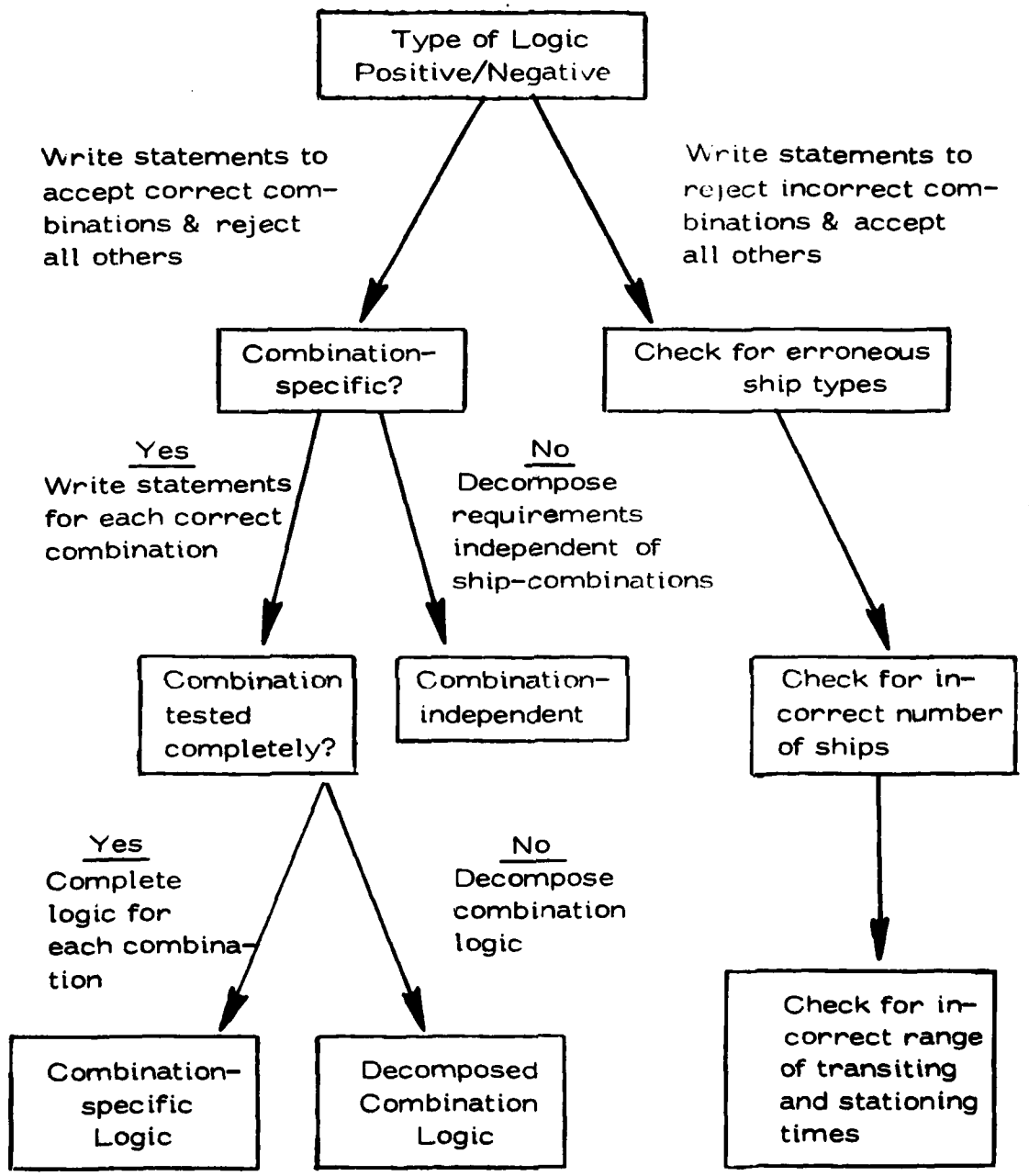


Figure 13. Strategy Considerations

Consider first the positive-logic path. Here, statements written to accept correct ship-combinations can be tailored to each acceptable combination or else the logic of the requirements can be generalized. For example, consider the logic required by problem #93, as shown in Figure 14. Three distinct combinations of ship types are specified in which each ship type may have specific transiting- and stationing-time requirements. The combination-specific approach results in separate, specific logic statements for each of those three correct ship-combinations. A requirements-decomposition approach, on the other hand, could result in the separate statement that two submarines (SS) are required in all combinations with the remaining ships specified in additional, separate statements. The requirements-decomposition approach might also result in statements concerning the logic of the aircraft carriers, e.g., that the sum of CVA's plus CVAN's must equal 2, stated independently.

If a combination-specific strategy is employed rather than a combination-independent strategy, a choice remains between specifying each combination completely ship by ship, or decomposing it into statements separately concerning the number of ship types, transiting times, and stationing times.

Now, consider the negative-logic path. Here, statements were written to reject ship-combinations that contained the following errors:

1. Extraneous ship types,

2. Incorrect number of correct ship types, and

3. Incorrect range of stationing and/or transiting times for correct ships types.

It must be noted that a ship selection logic can use one path starting from the top (see Fig. 13) to one of the various end points or it can use a mixed strategy. An example of a mixed strategy would be the specification of acceptable ship types for each ship-combination (a combination-specific logic) and the blanket rejection of erroneous ships (a combination-independent logic).

The specific strategies- and techniques/style-factors and their assigned codes for the Fortran subroutines were:

1. The ships needed for the task force are:

   o 2 Attack Aircraft Carriers (CVA or CVAN),

   and

   o 2 Submarines (SS)

This task force criteria specified 3 combinations of ship types as follows:

   o 2 CVA and 2 SS

   or

   o 2 CVAN and 2 SS

   or

   o 1 CVA and 1 CVAN and 2 SS

Figure 14. Test Problem #93

Factor	Code Assigned
1. Combination-specific logic	1 if used, otherwise 0
2. Decomposed-combination logic	1 if used, otherwise 0
3. Combination-independent logic	1 if used, otherwise 0
4. Other strategies (e.g. mixed)	1 if used, otherwise 0
5. Reject erroneous ship types	1 if used, otherwise 0
6. Reject incorrect transiting and stationing times	1 if used, otherwise 0
7. Comments	1 if one or more comments, otherwise, 0
8. Parallel construction of IF statements	0 if no IF statements have parallel construction; 1 if at least one pair, but not all IF statements have a parallel construction; 2 if all IF statements have a parallel construction
9. Use of addition for logical "OR" statements	1 is used, otherwise 0
10. Do Loop	1 if used, otherwise 0
11. Total number of factors used	Number

Note that parallel construction of IF statements refers to IF statements on adjacent lines, spaced so that variables and logical operators such as ".AND.", or ".OR." are arranged in columns to give a neat, orderly presentation. Note, also, that addition can be used to implement multiple "OR" functions. For example, the statement "The number of CVA's and CVAN's must equal 2" can

replace the logic statement "CVA. EQ.2 .OR. CVAN .EQ.2 .OR. (CVA .EQ.1 .AND. CVAN .EQ.1)".

An analysis of the frequency of use of the individual strategies was conducted. A correlation analysis among the eleven strategy factors was conducted. Finally, univariate and step-wise linear multivariate regression analyses were run using the eleven measures of programming strategy for the independent variables and the three performance measures (P_C, P_{IC}, and P_T) for the dependent variables.

Results. Table 35 gives the usage means and standard deviations of the codes for each strategy factor. "Combination-independent" strategy was used in 65 % of the subroutines and thus was the most favored strategy. "Decomposition of logic requirements" was used in 22.5 % of the subroutines and was the second most common strategy. "Comments," "combination-specific", "other," and "Do logic" were used in less than 10 % of the subroutines. Parallel construction of IF statements had a mean code value of 1.11, indicating that a moderate degree of parallel construction was typical. Finally, the number of strategy and style factors used in the average subroutine was 2.8, indicating that almost 3 factors were typically used together in an "average" subroutine.

Table 36 gives the cross-correlation coefficients for the strategy and programming-style factors. Note that "combination-independent strategy", the one most frequently used, was not highly positively correlated with any other factor including "comments" and "total number of factors used" suggesting that, when combination-independent strategy was used, it was typically used alone and without comments. Likewise, the strategy "decomposition of combination logic", the second most frequently used strategy, was apparently frequently used alone and without comments. Conversely, the negative-logic factors "reject incorrect transiting and stationing times" and "reject extraneous ship types" were frequently used together (the correlation value was .476, although the correlation had been expected to be closer to 1.0) along with "addition for logical OR", and "comments".

Tables 37, 38, and 39 provide the results for the univariate and multivariate regressions using P_C, P_{IC}, and P_T, respectively, for the dependent variables. Table 37 shows that significantly

Table 35

Strategy Use

	Mean *	Standard Deviation
Comments	.067	.250
Parallel Construction of IF Statements **	1.117	.769
Combination-Specific	.058	.235
Decompose Combination Logic	.225	.419
Reject Incorrect Transiting and Stationing Times	.308	.464
Combination-Independent	.650	.479
Other	.033	.180
Use Addition for Logical "OR"	.308	.464
Reject Extraneous Ship Types	.317	.467
Use DO Loop	.083	.278
Total Number of Factors Used **	2.808	1.272

* Mean number of times the strategy, or programming-style, factor was used.

** Mean value of assigned score (see code descriptions in Method section)

Table 36

Strategy Correlations

	2	3	4	5	6	7	8	9	10	11
1 Comments	-.172	.076	-.144	.328	.056	-.050	.256	.177	.040	.436
2 Parellel Construction of IF Statements		-.131	-.082	-.220	.043	.093	.040	-.151	-.007	.100
3 Combination Specific			-.134	-.089	-.339	-.046	.065	.060	.182	.038
4 Decompose Combination Logic				.029	-.525	-.100	.159	-.109	-.018	.066
5 Reject Incorrect Transiting & Stationing Times					-.115	-.124	.297	.476	-.071	.585
6 Combination Independent						-.253	-.115	.011	.032	.068
7 Other							-.124	-.126	-.056	-.118
8 Use Addition for Logical "OR"								.321	.256	.699
9 Reject Extraneous Ship Types									.184	.669
10 Use DO Loop										.379
11 Total Number of Factors (Above) Used										

Table 37

Correlation/Regression Analyses:
Strategy Factors vs.
Probability of Accepting
Correct Ship-Combinations (P_C)

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P≤
		Univariate Regressions			
Combination Specific	.178	.341	3.2	1.96	.05
Combination Independent	-.175	-.165	3.1	-1.93	.05
DO Loop	-.125	-.203	1.6	-1.37	.1
		Multi-variate Regression			
Combination Specific	.178	.435	10.3	2.49	.01
DO Loop	-.125	-.309		-2.04	.025
Decompose Combination Logic	.105	.133		1.36	.1
Other	.100	.321		1.42	.1
Use of Addition for Logical OR	.098	.124		1.37	.1

Analysis of Variance Table*

Source	Sum Squares	df	Mean Squares	F-Ratio	P≤
Regression	2.48	5	.497	2.61	.05
Residue	21.72	114	.190		

*Analysis of Variance Table for the Multi-Variate Regression

Table 38

Correlation/Regression Analyses:
Strategy Factors vs.
Probability of Rejecting
Incorrect Ship-Combinations (P_{IC})

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P≤
	Univariate Regressions				
Comments	.139	.154	1.9	1.53	.10
Reject incorrect transiting & stationing times	.177	.106	3.1	1.95	.05
Combination-independent	.162	.094	2.6	1.78	.05
Other	-.286	-.440	8.2	-3.23	.001
Use addition for logical "OR"	.190	.113	3.6	2.09	.025
Reject extraneous ship types	.342	.203	11.7	3.95	.0005
Use DO loop	.193	.193	3.7	2.13	.025
Total No. of factors used	.287	.062	8.2	3.25	.001
	Multi-variate Regression				
Reject extraneous ship types	.342	.164	22.9	3.26	.001
Other	-.286	-.403		-3.13	.001
Combination-specific	-.099	-.203		-2.04	.025
Use DO Loop	.193	.157		1.84	.05
Decompose combination logic	-.104	-.079		-1.43	.1

Analysis of Variance Table *

Source	Sum Squares	df	Mean Squares	F Ratio	P≤
Regression	2.109	5	.421	6.78	.01
Residue	7.088	114	.062		

*Analysis of Variance Table for the Multi-Variate Regression

Table 39

Correlation/Regression Analyses:
Strategy Factors vs.
Probability of Correctly Accepting or
Rejecting a Ship-Combination (P_T)

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	$P \leq$
Univariate Regressions					
Comments	.169	.106	2.9	1.86	.05
Reject incorrect transiting & stationing times	.238	.080	5.6	2.65	.005
Other	-.264	-.230	7.0	-2.97	.0025
Use addition for logical "OR"	.317	.107	10.1	3.63	.0005
Reject extraneous ship types	.401	.135	16.1	4.75	.0005
Use DO Loop	.123	.069	1.5	1.34	.1
Total No. of Factors Used	.385	.047	14.8	4.52	.0005
Multi-variate Regression					
Reject extraneous ship types	.401	.105	24.0	3.65	.0005
Other	-.264	-.175		-2.45	.01
Use Addition for Logical "OR"	.317	.064		2.22	.025

Analysis of Variance Table*

Source	Sum Squares	df	Mean Squares	F-Ratio	$P \leq$
Regression	.707	3	.235	12.17	.01
Residue	2.243	116	.019		

*Analysis of Variance Table for the Multi-Variate Regression

associated with P_C in univariate regressions were the "combination-specific" and "combination-independent" strategies as well as "Do loop". "Combination-specific" was the only significant factor positively correlated with P_C. In the multivariate analysis, "combination-specific", as well as "decompose combination logic," "other", and "use of addition for logical OR" were significantly and positively correlated with P_C. "DO loop" was significant, but was negatively correlated with P_C.

The positive correlation of the "combination-specific" strategy factor emphasizes the reliability of combination-based logic. Its ability to account for only 3.2% of the variance (in the univariate analysis) may be due to its use in only 6 % of the subroutines. In contrast, "combination-independent" strategy, which was negatively correlated with P_C and accounts for only 3.1 % of the variance, was used in 65% of the sub-routines.

Table 38 reveals that many factors were significantly correlated with P_{IC}. "Reject extraneous ship types" appears to have been the most important factor followed by "total number of factors used" and "other". "Reject extraneous ship types" was apparently used with a greater degree of success then was "reject incorrect transiting and stationing times". "Comments", "use of addition for logical OR", and "use DO loop" were also used successfully for rejecting incorrect ship-combinations.

The multivariate regression presented in Table 38 reveals, in addition to the results reported above, that combination oriented strategies, namely "combination-specific" and "decompose combination logic", did not contribute positively to the successful rejection of incorrect ship-combinations. These strategies are classified as positive-logic strategies since they are used to directly specify correct combinations -- the remaining combinations being rejected as incorrect. Apparently, this was not completely accomplished, so that frequent errors occurred in rejecting incorrect combinations.

Table 39 gives the results of the univariate and multivariate analyses which used P_T as the dependent variable. Results of the univariate analyses were similar to those for P_{IC}, except that the strategy "combination-independent" was not found to be significant. The remaining factors found to be significant for P_{IC}

were also found to be significant for P_T. Note that "other" and "use Do loop" showed increases in their correlations for the same comparison. This reflected the common, successful use of those strategy factors to both accept correct and reject incorrect combinations.

Analysis 3: Demographic Factors and Performance

Purpose. To determine the relationships among the participant demographic factors and the effect of those factors on the completeness and accuracy of the Fortran IV code written by the participants.

Method. A correlation analysis was performed for the independent demographic variables. Univariate and step-wise multivariate regression analyses were run using the demographic and experience factors for independent variables and the three performance measures (P_C, P_{IC}, and P_T), described above, for the dependent variables.

The demographic data were collected from the biographical forms completed by each participant. The following categories were used:

1. Age,
2. Years-of-experience,
3. Years of-advanced-education (beyond high school),
4. Number of programs written,
5. Number of computer systems used,
6. Number of programming languages used,
7. Number of programming areas involved in among the following:
   a. Data entry
   b. Production control
   c. Operations
   d. Applications programming
   e. System programming
   f. System analysis
   g. Data base administration
   h. Data communication
   i. Other

Results. Results of the correlation analysis among the independent variables are shown in Table 40. "Age" and "Years of experience" were highly correlated and cannot be considered independent factors. Further, the "number of programming areas involved in" was moderately correlated with "age", "years-of-experience", "number of programs written", and "number of systems used". The "number of systems used" and the "number of languages used" were moderately correlated.

Table 40

Correlation of Demographic Variables

	Exp.	Educa.	No. of Prog.	No. of Systems	No. of Lang.	No. of Areas
Age	.851	-.009	.322	.399	.066	.574
Experience		-.050	.436	.419	.139	.691
Education			.068	-.162	.106	-.153
No. of Prog.				.274	.325	.512
No. of Systems					.446	.595
No. of Lang.						.268

Tables 41, 42, and 43 provide the results of the regression analyses for dependent variables P_C, P_{IC}, and P_T, respectively. All demographic variables were included in the tables independent of their statistical significance, because the inconclusive results obtained for some variables were unexpected. For instance, note that "years-of-higher-education", contrary to expectation, was not found to be a statistically significant factor in univariate regressions. Further, note that the "number of computer systems used" and the "number of programming languages used" were highly correlated with P_C and, together with "years-of-higher-education", predicted 34% of the P_C score variance. Also, these same factors, while positively correlated with P_C were negatively correlated with P_{IC} and P_T.

Table 41

Correlation/Regression Analysis: P_C

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P≤
Univariate Regressions					
Age	.126	.0033	1.6	.671	NS
Year's Exp.	.148	.0051	2.2	.791	NS
Education	.218	.0282	4.8	1.183	NS
# Programs	.090	.0001	.8	.478	NS
# Systems	.457	.0523	20.9	2.719	.01
# Prog. Lang.	.441	.0663	19.4	2.597	.01
# Prog. Areas	.269	.0303	7.3	1.48	NS
Multi-variate Regression					
Education	.218	.0332	34.0	1.55	.1
# Systems	.457	.0449		2.14	.025
# Prog. Lang.	.441	.0359		1.31	.1

Analysis of Variance Table *

Source	Sum Square	df	Mean Square	F-Ratio	P ≤
Regression	.512	3	.170	4.468	.05
Residue	.993	26	.038		

* Analysis of variance Table for multi-variate regression

Table 42

Correlation/Regression Analysis: P_{IC}

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P ≤
Univariate Regressions					
Age	-.464	-.0100	21.5	-2.773	.005
Year's Exp.	-.350	-.0099	12.2	-1.974	.05
Education	.022	.0023	0.0	.117	NS
# Programs	-.223	-.0002	5.0	-1.209	NS
# Systems	-.477	-.0442	22.7	-2.868	.005
# Prog. Lang.	-.111	-.0135	1.2	-.591	NS
# Prog. Areas	-.541	-.0493	29.2	-3.402	.001
Multi-variate Regression					
Age	-.464	-.0137	40.4	-2.19	.025
Year's Exp.	-.350	.0169		1.82	.05
# Areas	-.541	-.0537		-2.81	.005

Analysis of Variance Table *

Source	Sum Square	df	Mean Square	F-Ratio	P ≤
Regression	.400	3	.1333	5.87	.001
Residue	.590	26	.0226		

*Analysis of variance Table for multivariate regression

Table 43

Correlation/Regression Analysis: P_T

Independent Variable	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P≤
		Univariate Regressions			
Age	-.444	-.0059	19.7	-2.623	.01
Year's Exp.	-.304	-.0053	9.2	-1.687	.05
Education	.148	.0096	2.2	.792	NS
# Programs	-.208	-.0001	4.3	1.127	NS
# Systems	-.250	-.0143	6.3	1.368	.1
# Prog. Lang.	.145	.0109	2.1	.776	NS
# Prog. Areas	-.438	-.0247	19.2	-2.580	.01
		Multi-variate Regression			
Age	-.444	-.0087	38.3	-2.170	.025
Year's Exp.	-.304	.0103		1.732	.05
# Prog. Lang.	.145	.0189		1.528	.1
# Areas	-.438	-.030		-2.397	.01

Analysis of Variance Table *

Source	Sum Square	df	Mean Square	F-Ratio	P ≤
Regression	.145	4	.0362	3.88	.05
Residue	.233	25	.0093		

*Analysis of variance Table for multivariate regression

P_{IC}, the probability of rejecting incorrect ship combinations, was predicted by a different set of variables from those that predicted P_C. They were "age," "years-of-experience," and "number of programming areas." All three were negatively correlated with P_{IC} and together explained 40% of the variance.

The total probability, P_T, was predicted by "age," "years-of-experience," "number of programming languages used," and "number of programming areas involved in." Only "number of programming languages used" was positively correlated with P_T. Note that there was a reduced correlation of these factors with P_T compared to their respective correlations with P_{IC} and P_C, suggesting that their effects on P_{IC} and P_C tended to cancel each other in the prediction of P_T.

Analysis 4: Relationships Between Compiler-Detected Errors and Performance Scores.

Purpose. To determine what relationships existed, if any, between the compiler-detected errors (many of which, by reflecting clear intent, were corrected and therefore did not directly result in a performance decrement) and performance scores.

Method. A correlation analysis and univariate and stepwise multivariate linear regression analyses were run using the frequency of 14 types of errors as the independent variables and the three performance measures as the dependent variables. The 14 error types consisted of 13 that were detected by the compiler plus one additional variable, which was equal to the sum for each participant of his non-zero error-types. This latter variable was the count of the number of error-types that occurred in the participant's problem solutions i.e. in his subroutines. The error-types were the following:

Index Number	Error-Type
1.	Dotted-keyword error,
2.	Variable exceeds 6 characters,
3.	Continuation mark not in Col. 6,
4.	Blank line,
5.	Misspelled word,
6.	Parenthesis error,
7.	Missing statement,

8.	Statement starts in Col. 1-6
9.	Statement exceeds 80 characters,
10.	Statement syntax error,
11.	Wrong variable,
12.	Array-index error,
13.	Fixed point/floating point error,
14.	Sum of non-zero error types (1-13).

Results. The correlations between each error-type and the three performance scores are shown in Table 44. Note that each error-type strongly correlated with P_C, i.e. with a correlation greater than .1 or less than -.1, had a reverse polarity in its correlation with P_{IC}. Apparently, although the correlations in general were weak, there was a tendency for some factors to be positively correlated with P_C and negatively correlated with P_{IC}, or vice versa.

Results for the univariate regressions and a step-wise multivariate regression for each of the dependent variables P_C, P_{IC}, and P_T, are given in Tables 45, 46, and 47, respectively. The error-types in the multivariate regressions have been listed in the order of the amount of variance explained, starting with the greatest.

The regressions for P_C suggest that associated with a high probability of accepting correct ship-combinations were the absence of parentheses and syntax errors, and the presence of variables exceeding six characters. (Note: excessively long variables were corrected before the subroutine test.) In contrast, the variables associated with the rejection of incorrect ship-combinations were the absence of errors in the placement of continuation marks, the absence of statements begun in Column 6 or less, and the absence of excessively long variables. Also associated with the rejection of incorrect ship-combinations were syntax and parenthesis errors.

While the percent of variance explained by the regressions was modest -- 14% for P_C and P_{IC} and only 6% for P_T -- the results suggest that there may be error-tendencies or associations that might be used to guide and refine the testing of a program based on initial compiler-detected errors.

Table 44

Correlation of Programming Error - Types
With Performance Measures

Programming Error Type	Correlation With P_C	Correlation With P_{IC}	Correlation With P_T
Variable exceeds 6 characters	.152	-.184	-.111
Line exceeds 80 characters	.077	-.023	.042
Cont. mark in wrong column	.045	-.214	-.215
Wrong variable	.040	-.026	.008
Misspelled word	-.003	-.034	-.051
Statement start in Col. 1-6	-.038	-.074	-.130
Blank lines	-.039	.057	.052
Array-index error	-.053	-.011	-.060
Missing statement	-.061	-.077	-.141
Floating point/ fixed pt. error	-.066	.046	-.012
Dotted-keyword error	-.080	-.005	-.081
Parenthesis error	-.180	.110	-.029
Total number of error types	-.238	.016	-.178
Syntax error	-.265	.168	-.033

Table 45

Compiler-Detected Programming Error-Types vs.
Probability of Accepting Correct Ship Combinations (P_C)

Error Type	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P≤
		Univariate Regressions			
Syntax Error	-.265	-.050	7.0	-2.99	.0025
Number of Error Types	-.238	-.071	5.6	-2.65	.05
Parentheses Error	-.180	-.035	3.2	-.199	.025
Variable Exceeds 6 Char.	.152	.219	2.3	1.67	.05
		Multi-variate Regression			
Syntax Error	-.265	-.044	14	-2.69	.005
Parentheses Error	-.180	-.053		-2.56	.01
Wrong Variable	.040	.069		1.80	.05
Variable Exceeds 6 Char.	.152	.196		1.57	.1

Table 46

Compiler-Detected Programming Error-Types vs.
Probability of Rejecting
Incorrect Ship-Combinations (P_{IC})

Error Type	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P≤
		Univariate Regressions			
Cont. Mark in Wrong Column	-.214	-.021	4.6	-2.38	.01
Variable Exceeds 6 Char.	-.189	-.167	3.6	-2.09	.05
Syntax Error	.168	.019	2.8	1.85	.05
		Multi-variate Regression			
Cont. Mark in Wrong Column	-.214	-.024		-2.71	.005
Syntax Error	.168	.025		2.38	.01
Variable Exceeds 6 Char.	-.189	-.152	14.1	-1.97	.05
Statement Starts in Col. 6 or Less	-.074	-.014		-1.46	.1
Paren-theses Error	.110	.013		1.29	.1

Table 47

Compiler-Detected Programming Error-Types vs. Probability of Correctly Accepting or Rejecting a Ship Combination (P_T)

Error Type	Correla. Coeff.	Regre. Coeff.	% Variance Explained	t-Value	P≤
		Univariate Regressions			
Cont. Mark in Wrong Column	-.215	-.012	4.6	-2.39	.01
No. of Error Types	-.178	-.018	3.2	-1.96	.05
Missing Statements	-.141	-.034	2.0	-1.55	.1
Statement Starts in Col. 6 or Less	-.130	-.007	1.7	-1.41	.1
		Multi-variate Regression			
Cont. Mark in Wrong Column	-.215	-.010	6.2	-1.95	.05
No. of Error Types	-.178	-.013		-1.42	.1

Discussion Experiment #6

General Comments on Coding Strategy

Which strategy is appropriate for selecting ship-combinations (or in general for selecting any set of factors) depends on the nature of the problem being solved. (In the discussion that follows, although we use the ship-selection terminology developed for the experiment, the concepts apply to selection logic in general). If, for instance, there are only a few ship-combinations that are acceptable and many that are to be rejected, then using positive logic requires fewer statements than using negative logic. Conversely, if there are many ship-combinations to be accepted and only a few to be rejected, then negative logic identifying the specific combinations to be rejected requires the fewer statements.

An example of a positive-logic, combination-specific strategy is the following. Suppose an acceptable ship-combination for a task force is 1 non-nuclear carrier (CVA) and 2 non-nuclear submarines (SS). If there are 9 possible ship types on the ship-list, then a positive -logic Fortran IV statement specifying the desired combination would be:

```
If (CVA .EQ. 1 .AND.
   CVA .EQ. 0 .AND.
   CA  .EQ. 0 .AND.
   CG  .EQ. 0 .AND.
   CGN .EQ. 0 .AND.
   DD  .EQ. 0 .AND.
   SS  .EQ. 2 .AND.
   SSN .EQ. 0 .AND.
   AO  .EQ. 0 .AND.) ACCEPT = .TRUE.
```

Here all the variables are tested individually, since each variable, i.e. each ship-type, is represented by an integer, which may be zero.

But it is not necessary to use only positive or negative combination-specific logic. If a part of, or all of, the ship-combinations to be rejected or accepted can be specified in the same manner for all combinations, then use of a combination-independent strategy can significally reduce the specification-logic complexity.

For instance, if, in the problem given above, all combinations require only CVA's and SS's with all other ship types equaling zero, then one negative-logic statement could be written to reject all combinations with non-zero CVAN, CA, CG, CGN, DD, SSN, or AO's. The result would be a less complex positive-logic statement for specifying those combinations which are acceptable.

Based on the above discussion, it should be apparent that combination-specific positive or negative logic works at a level of detail similar to that of example-solutions -- i.e. each combination is treated both explicitly and independently of the other combinations. If the participants in Experiment 6 had used a combination-specific strategy, we might have expected the same performance as obtained in Experiment 1, where example-solutions to the same problems were developed. But in Experiment 6 participants frequently tended to use combination-independent logic (in 65% of the experiment problems). This combination-independent logic employed statements that were intended to be valid for all ship-combinations. As a result, when correctly developed, the specification of numerous ship-combinations could be simplified into a few combination-independent logic statements. This simplification may account for the fact that problem-complexity was not a significant factor in the experimental results. The poor performance obtained in the experiment suggests that errors were made in translating the problem specifications into combination-independent logic. Thus, either combination-specific logic should have been used or an aid, such as those used in Experiments 3 & 4, adapted to organize statements, should have been developed and made available.

The fact that the strategy-measures used to analyse Experiment 6's data did not have the same performance-predictive power as the strategy-measures used for Experiments 1-5 suggests that sensitive measures were not identified. In fact, the strategy-measures for Experiment 6 were not moment-to-moment measures, but together represented only a way of classifying overall strategy. Obviously, a moment-to-moment measure for computer programming (designing and coding) should be developed.

Performance Measures

The relationship between P_T and P_C & P_{IC} provides insight into a serious problem in classifying programming errors.

P_T can be considered one type of error, and P_C and P_{IC} can be considered to be the elements in a decomposition of P_T. (See Equation 6). Yet, some of the strategy and programming-technique factors that predicted P_C in a multivariate regression were different from those that predicted P_{IC}. And, those that were the same in both regressions could show different correlation polarities. For example, "Do loop" was negatively correlated (-.125) with P_C, but positively correlated (.193) with P_{IC}. As a result, its correlation (.123) with P_T was reduced, and it did not appear in the multivariate regression for P_T. A similar result occurred for "combination-specific," and the same tendency occurred for "other," although "other" did not drop out of the regression for P_T. Similar results occurred when the demographic factors were used as the independent variables. For instance, the "number of programming languages used" and the "number of operating systems used" were positively correlated with P_C, negatively correlated P_{IC}, and, as a result, only moderately correlated with P_T. The point is that combining error categories (such as "logical," "Data," etc.) as is usually done for ease of data-classification and -collection purposes may lead to serious analytical problems. For example, combining all errors thought to be "logical" into one category -- e.g. the errors resulting in the erroneous assignment of a ship-combination (measured by P_T) could be termed "logical" errors - can obscure predictive relationships that actually exist in portions of the error category. If, for instance, we had only used the category of errors measured by P_T, we would not have discovered that "number of systems used" was an important factor in predicting P_C, even though P_C was part of P_T.

We conclude, then, that error categories should be selected with regard to their predicability. And two categories should be combined only when their prediction (regression) equations use the same independent variables (are homologous). Further, study of the mechanisms or conditions that result in errors may permit definition of fundamental categories of errors.

Demographic Factors

Prediction of P_{IC}, P_C, and P_T using demographic factors as the independent variables led to results somewhat similar to those found for the same demographic factors in Experiments 1-5. The "number of programming languages used"

and "number of operating systems used" were positively correlated and were significant predictors of P_C. Also, the "number of programming languages used" was a significant predictor of $P_{\_}$. However, both were negatively correlated with P_{IC}. Apparently, there was a fundamental difference in experience that differently affected accepting correct items and rejecting incorrect items.

Analysis and Comparison of Example-Solution and Program-Code Errors

The purpose of the analysis presented in this section is to compare the errors produced when specification of a problem solution is accomplished in two different ways: by example-solutions and by program code. Our underlying purpose is to gain insight into why, under certain conditions, one problem-solution specification method might be superior to the other.

Two types of errors were analyzed. One type, termed an "error-of-omission", refers to an error that results in a failure to accept a correct entity (ship-combination). When specifying a problem solution with example-solutions, an error-of-omission can be directly traced to a failure to enter an example of a suitable entity (e.g. ship-type) or to an example-solution that fails to properly establish the desired range of a variable (in the experiment problem the desired range of transiting and stationing times). There is a one-to-one correspondence between an error-of-omission and the resulting logic error when using example-solutions to specify a problem solution. But, when using program code to specify the problem solution, there no longer exists a simple one-to-one relationship between an error-of-omission and the resulting logical error. Instead an erroneous statement in the program code may result in multiple logic errors. In either case, the probability of an error-of-omission (P_{E-O}) equals one minus the probability of accepting an acceptable entity (P_C). That is:

$$P_{E-O} = 1 - P_C \qquad (7)$$

The second type of error considered in this section is termed an "error-of-commission." When example-solutions are used to specify a problem solution, an error-of-commission

corresponds to the entry of an incorrect example into the processor which is then treated by the processor as a correct example. An error-of-commission results in erroneously accepting incorrect entities (e.g. ship-combinations). Again, a one-to-one correspondence between an error-of-commission and the corresponding logical error does not exist when program code is used for specifying a problem solution. In either case, the probability of an error-of-commission (P_{E-C}) equals one minus the probability of rejecting an incorrect entity (P_{IC}). That is:

$$P_{E-C} = 1 - P_{IC} \qquad (3)$$

Analysis 1: Comparison of Errors-of-Omission for Example-Solutions vs. Program Code

Purpose. To determine whether there was a significant difference between the example-solution and the program-code methods of specifying problem solutions for the probability of an error-of-omission. (P_{E-O})

Method. The probability of an error-of-omission for each of the nine experiment cells of Experiment 1 (where example-solutions were used) was compared to the same probability for the corresponding cells in Experiment 6 (where Fortran IV code was used). Both participant groups in the analysis were experienced programmers selected using identical criteria. A t-test was used to determine the statistical significance of any differences.

Results. Table 48 gives the probability of an error-of-omission (P_{E-O}) for Experiment 1 and Experiment 6. As indicated, the only cell with a significant difference was $A_1 B_3$, which corresponds to the least-complex problem in both experiments and to processor-complexity Level B_3 in Experiment 1 and to Mission-Type 3 in Experiment 6. In each experiment, participants working with this cell had to input the maximum amount of information for that problem level. This cell also provides the only instance in which participants clearly omitted fewer correct entries when using example-solutions than when using program code to specify a problem solution. Otherwise, as shown by the insignificant difference in the grand means for both experiments, no superiority

Table 48

Comparison of the Probability of an Error-of-Omission* for Example-Solutions vs. Program Code

Experiment #1 Data - Programmers

Problem-Complexity Levels

Processor-Complexity Level	A_1	A_2	A_3
B_1	.167 (.833)	.367 (.643)	.342 (.658)
B_2	.325 (.675)	.381 (.619)	.479 (.521)
B_3	.481 (.519**)	.702 (.298)	.668 (.332)

Grand Mean = .440 (.560)

CELL CODE:

P_{E-0} (P_C)

Experiment #6 Data-Programmers

Problem-Complexity Levels

Mission Types	A_1	A_2	A_3
1	.246 (.754)	.237 (.763)	.335 (.665)
2	.511 (.489)	.359 (.741)	.476 (.524)
3	.859 (.141**)	.771 (.229)	.704 (.296)

Grand Mean = .490 (.510)

(Reproduced from Table 30)

* Probability of an Error-of-Omission (P_{E-0}) = 1- probability of accepting a correct entity (P_C).

**Difference statistically significant t (18) = 2.61, $P < .01$.

could be demonstrated for one method over the other with respect to errors-of-omission.

Note, too, that when example-solutions were used, performance as measured by P_C degraded as both problem-complexity (Factor A) and the amount of information required per ship-combination (Factor B, or processor-complexity) were increased, but that, when program code was used, problem-complexity did not result in a performance degradation. Instead, the amount of information required (reflected by mission-type, which was intended to be equivalent to Factor B in Experiment 1) became the significant factors affecting performance.

Discussion. Apparently, for the range of problem-complexities studied in this experiment series, there was little difference in the effect of problem-complexity on errors-of-omission between the two methods of specifying problem solutions. However, the two methods contrasted markedly when the information requirement (Factor B) was increased for the least complex problem (A_1). Experiment 1, in which example-solutions were used, showed only a moderate reduction in performance; Experiment 6, in which program code was written, showed a more drastic performance degradation. This result indicates that, when problem-complexity (Factor A) is low and the amount of information required/item (Factor B) is high, specification with example-solutions may be superior to writing program code.

The data in Table 48 for program code suggest that the amount of information required per item can significantly increase the frequency of errors-of-omission. Apparently, to increase performance, problem-solution designs should limit the amount of information required per item when program code is being written.

Analysis 2: Comparison of Errors-of-Commission for Example-Solutions vs. Program Code

Purpose. To determine whether there was a significant difference between the example-solution and the program-code methods of specifying problem solutions for the probability of an error-of-commission (Γ_{E-C}).

Method. Since the ANOVA for P_{IC} for Experiment 6 (Table 31) indicated that the experiment factors (problem-complexity and mission-type) had no significant effects on P_{IC}, a grand mean value for P_{E-C} was calculated. Since the same findings were obtained for Experiment 3, where various feedback-aid levels were used and where processor-complexity level B_2 was maintained over all the experiment cells, a grand mean for P_{E-C} was likewise calculated. For Experiment 1, however, in which performance significantly degraded for problem level A_3, the most complex, a mean was calculated only for data pertaining to problem levels A_1 and A_2.

Results. As shown in Table 49, the mean probability for an error-of-commission for the six cells of Experiment 1 that excluded problem-complexity level A_3 was .07. Apparently, errors-of-commission increase when attempting to generate example-solutions for problems whose complexity, as measured by Halstead's E Metric (See Connelly et al., 1981), equals or exceeds 20,821 (i.e. at or above level A_3, where the mean value of P_{E-C} was .35). In this correction, it is worth noting that the high value of .18 for cell A_1B_2 was traced to one individual who entered 9 example-solutions all of which were incorrect. Without this individual, the mean value of P_{E-C} was .04.

A clearer picture was obtained for the probability of an error-of-commission for participants engaged in the formation of example-solutions from the data for Experiment 3, also shown in Table 49. The grand mean value of P_{E-C} for Experiment 3, for all cells, was .03.

Finally, for Experiment 6, in which participants wrote program code to specify solutions, the grand mean value of P_{E-C} was .177. The difference in the grand mean values between Experiment 3 and Experiment 6 was found to be significant, with $\underline{t}(178) = 143$, and $p \leq .0005$.

Discussion. Comparison of the mean probabilities for Experiments 1, 3, and 6 strongly suggests that the rate of errors-of commission can be substantially reduced by using example-solutions, rather than Fortran IV program code, for specifying problem solutions. The 3 to 4 percent rate of errors-of-commission for example-solutions compares favorably with the 18 percent rate for program code.

Table 49

Comparison of the Probability of an Error-of-Commission* for Example-Solutions vs. Program Code

Experiment #1 Data - Programmers

Problem-Complexity Level

Processor Complexity Level	A_1	A_2	A_3
B_1	0.0 (1.00)	.06 (.94)	.22 (.68)
B_2	.18 (.82)	.03 (.97)	.43 (.47)
B_3	0.0 (1.00)	.05 (.95)	.41 (.49)

Mean for cells excluding A_3 = .07 (.93)

CELL CODE:

P_{E-C} (P_{IC})

Experiment #3 Data - Programmers (Processing Level B_2)

Problem-Complexity Level

Feedback-Aid Levels	A_1	A_2	A_3
F_1	0.0 (1.00)	0.0 (1.00)	.02 (.98)
F_2	.04 (.96)	.02 (.98)	.07 (.93)
F_3	.04 (.96)	.03 (.97)	.01 (.99)

Grand Mean = .03 (.97)

*Probability of an Error-of-Commission (P_{E-C}) = 1- probability of rejecting an incorrect item (P_{IC})

Table 49 (Concluded)

Comparison of the Probability of an Error-of-Commission* for Example-Solutions vs. Program Code

Experiment #6 Data (P_{IC}) - Programmers

Mission Types	A_1	A_2	A_3
1	.252 (.748)	.192 (.808)	.295 (.705)
2	.112 (.866)	.169 (.831)	.137 (.863)
3	.092 (.908)	.172 (.828)	.175 (.825)

Grand Mean = .177 (.823)

Comparison of the Mean Value of P_{E-C} and P_C

1. Mean for Experiment 1 over the 6 cells not including problem-complexity level A_3 = .07 (.93)
2. Mean for Experiment 1 over the same cells, but not including one participant whose performance was unusually poor. = .04 (.96)
3. Grand Mean for Experiment 3 = .03 (.97)*
4. Grand Mean for Experiment 6 = .177 (.823)*

*Statistically significant difference
$t(178) = 143$, $P \leq .0005$

The rate of errors-of-commission when using example-solutions increased sharply, based on Experiment 1's data (Table 49), at a problem-complexity level near 20,821, as measured by Halstead's E Metric. Given a suitable environment of feedback-aid, however, as in Experiment 3, this degradation of performance did not occur. Furthermore, cells in Experiment 3 with common experimental conditions in Experiment 1, i.e. at processor-level B_2 and feedback-aid level F_1, did not exhibit a degradation of performance for the most complex problem. Apparently, when using example-solutions, the instructions for using the feedback-aids and actual practice in using the aids on problems of low and intermediate complexity had the effect of reducing the rate of errors-of-commission on problems where, without the aids, performance had previously deteriorated.

We can form three hypotheses concerning the superior performance of the example-solution method:

1. It is working with examples and dealing with each individual combination of items one-at-a-time that results in a low rate of errors-of-commission.

2. It is the specification of each combination one-at-a-time that is important. Consequently, if computer programs were developed to specify each solution-combination one-at-a-time, the rate of errors-of-commission would be low.

3. The success of the example-solution method is due, in part, to the transformation of example-solutions from one logic-form into another, such as into the ship selection logic (SSL) or into several forms, such as the feedback-aids. Thus, it is the transformation of logic, which enables the user to view the problem in more than one way, that results in a low rate of errors-of-commission. Consequently, if program code entered by the user were transformed into a different logic-form (say, a different ordering of nested DO loops, or a complete specification of combinational forms, or structured vs. unstructured or combination-specific vs. combination-independent logic, etc.) and fed back to the user for approval, a low rate of errors-of-commission would be obtained.

These hypotheses are not alternative hypotheses - all could be true. We have strong evidence that the first hypothesis is true. If the second is true but not the third, program design and coding methods could be adapted to a more combination-dependent structure. And finally, if hypothesis 3 were found to be true, pre-compilation aids could be designed to convert the user's program code into another form (while maintaining the same program logic) for feedback to the user.

In a previous section regarding a programmer's knowledge of multiple programming languages and multiple operating systems, it was suggested that knowledge of, or ability to develop, alternative ways of viewing problems may be a key underlying concept. This same concept applied here to developing an alternative logic-form for program code suggests that hypothesis 3 might offer a means to reduce errors-of-commission not only in computer code but also in other software areas, such as requirements specifications and test-data generation.

CONCLUSIONS

1. Specifying a problem solution with example-solutions (working in conjunction with an inductive processor to convert examples into the implied logic) results in an error-of-omission rate approximately the same as the rate obtained when specifying a problem solution with Fortran IV program code. A significant improvement in the rate of errors-of-commission, however, does result when example-solutions, not Fortran IV program code, are used to specify problem solutions. (An error-of-omission is an error that results in failure to accept a correct entity. An error-of-commission results in failure to reject an incorrect entity). Thus, example-solutions can be used to specify accurate and complete problem solutions provided that suitable inductive logic is employed.

2. Without feedback aids, there appears to be an upper limit of problem complexity (20,821 as measured by Halstead's E Metric) above which large rates of errors-of-commission occur when using example-solutions to specify a problem solution. The use of appropriate feedback aids, however, either eliminates that limit or extends it beyond the complexity of the problems used in this series of experiments.

3. Feedback aids, to support the use of example-solutions to specify problem solutions, should include the logic implied by the example-solutions as well as suggested new example-solutions required to complete the logic patterns suggested by the existing set of example-solution. For instance, if example solutions AB, $\bar{A}B$, and $A\bar{B}$ have already been entered, the next solution which completes the logic pattern is $\bar{A}\bar{B}$.

4. Feedback-aids that include an ordered listing of all present solutions also support high performance. Even so, the predictive measure referred to above is preferred.

5. Both feedback-aid types referred to above assist in improving the rate of errors-of-omission for both programmers and non-programmers who would not perform well without the aids.

6. Performance measures designed to detect possible performance improvement with feedback aids should be relative measures which indicate the difference between performance on a common problem and each experimental problem. A relative measure is required because initial variance among participants is expected to be greater than variance improvement with the aids, i.e. users who need the aids but won't or can't use them (and therefore will exhibit poor performance) will not be affected by the aids. Only those who would not perform well without the aids and who use them will exhibit improved performance with the aids. Thus, the aids will influence performance of only a portion of the user population, and only a measure with a suitable sensitity (such as a relative measure) can detect such performance improvement.

7. The lack of a strong relationship between "years-of-higher-education" or "years-of-experience" and performance coupled with the strong relationships between performance and "number of computer languages" known and "number of operating systems" used suggests that education and experience should not be used as they have been in the past for hiring, promoting, determining salary levels, and making assignments. Instead, the number of computer languages known and the number of operating systems used, which are better performance predictors, should be used until the true underlying factors included in each are discovered.

8. Apparently, the depth of an individual's experience is not as important to performance as is the breadth of his experience. Evidence supporting this notion includes the result that the "number of programming languages used to code 10 or more programs" was not as significant a performance predictor as the "number of languages used to code at least 1 program." Further, the lack of significance of the "number of software application-areas experienced" supports the notion that, in themselves, software application areas do not broaden problem-solving capability.

9. There is strong evidence that the "number of programming languages" and "number of operating systems" used represent substantially different performance prediction factors, and do not represent the same performance-influencing factor(s) measured in two ways. Thus, there is apparently a set of underlying factors that affect performance embedded in these systems-

experience areas. There may exist other experience-areas that also affect performance.

10. A possible common, underlying, experience-related factor is the ability to view problems from alternative viewpoints -- the ability to develop alternative approaches to problems -- an ability that might be enhanced as more programming languages and operating systems are learned.

11. The performance-prediction capability of strategy measures, developed as moment-to-moment measures, not only clearly demonstrated that systematic strategies were used by successful participants, and not only led to the design of the feedback aids, but also convincingly demonstrated that moment-to-moment measures provided the sensitivity to explain considerable performance variance (approximately 60 % in Experiments 1 thru 4).

12. When modifying initially incorrect example-solutions, the probability (.934) of maintaining an initially correct example-solution was approximately the same as the probability of developing a new, correct example-solution. But, the mean probability of detecting and correcting an erroneous example-solution was low (.650). Further, the probability of detecting and correcting an erroneous example-solution was negatively correlated with the initial number of correct example-solutions. Apparently, the fewer the total number of errors the less was the likelihood of detecting a given error. This suggests that a method for increasing the probability of detecting errors is to seed errors, unknown to the individual reviewing the example-solutions (or computer program) but otherwise recorded, to increase the rate of error detection and therefore to increase the probability of detecting an unrecorded error.

13. Strategies used by successful individuals in revising example-solutions are apparently different from successful strategies used for developing original example-solutions. Further, aids designed to assist production of original solutions do not help in revising solutions. Successful strategies and aids for revision are yet to be determined.

14. Participants developing Fortran IV code tended to use combination-independent logic, i.e. attempted to form logic statements involving a subset of the problem variables that were true for all or at least most of the ship-combinations required. This approach could simplify the logic if implemented correctly; but, in these experiments, many errors were made converting the combination-specific logic of the problem statement to combination-independent logic.

15. Compiler-detected errors in Fortran IV code, as well as strategies- and programming-stlye factors, were used as independent variables in regression analyses and explained a moderate percent of the performance variance for errors-of-omission and-commission. Thus, elements of these factors, which are automatically measurable by computer processing of program code, could be used to estimate the number of undetected errors-of-omission and-commission in the program code.

16. Software error-categories are typically defined expressly to facilitate data collection and recording. However, our analysis of software errors showed that, when an error-category was decomposed into sub-categories (e.g. the total probability of error into errors-of-omission and -commission) the independent variables in the prediction (regression) equations changed. Some independent variables that were significant predictors of sub-categories of performance, such sub-categories as errors-of-omission or -commission, were not significant predictors of the combined performance factor "total logical error." This result was found for each set of independent variables studied: demographic, strategy, program-style, and compiler-detected errors. If only the combined error category "total logical error" had been used, relationships among (and possibly causes of) the component errors would not be known. Consequently, possible solutions would not be known. It is concluded, therefore, that software error-categories should be selected with regard to predictability as well as data collectability. Two categories should be combined only when their prediction equations are homologous, i.e. have the same independent variables. Possibly, a study of errors mechanisms or conditions that result in errors may permit definition of fundamental error categories, which would facilitate the collection, analysis, and correction of errors.

17. The superior performance (lower rate of errors-of-commission) achieved when using example-solutions and inductive processing to specify problem solutions over the performance achieved when using Fortran IV code may provide a basis for determing the underlying mechanism for that success and a means for incorporating that mechanism into program designing- and coding-aids. Apparently, superior performance was obtained either because each combination of the input variables was treated individually and/or because the example-solutions were transformed into another logic form - namely, the ship selection logic (SSL). If the former is a significant factor, then the feedback-aids described in this report should be adapted to program designing- and coding-aids. If the latter is a significant factor, then design- and code-aids should be developed to transform the logic provided by the user into another form which is then fed back to the user for his review. For instance, the user's program code might be transformed into a code with a different, but equivalent, logical structure. Alternatively, the transformation might present the program's equivalent logic to the user for review.

RECOMMENDATIONS FOR FURTHER RESEARCH

1. The strategy-measures used to analyse program code in Experiment 6 were not moment-to-moment measures. Instead, they were a classification of types of possible strategies. The predictive power of the measures was moderate compared to the strategy-measures used to evaluate performance in developing example-solutions. It is suggested that moment-to-moment strategy-measures be developed for both program-design and program-code tasks.

2. Feedback-aids designed to support development of original example-solutions were found not applicable to the revision of erroneous example-solutions. Since the use of example-solutions is an effective way to specify problem solutions and, further, may reveal ways of improving program design and code, successful revision-strategies should be identified, and revision-aids should be derived from those strategies.

3. The "number of programming languages" known and the "number of operating systems" used have been shown to be good predictors of performance both for developing example-solutions and for writing program code. It is suggested that the

ability to develop alternative approaches may be a common factor which may be enhanced by learning new languages and systems, and which may be a key performance factor. Further, it is suggested that breadth of experience is more important than depth in specifying complete and accurate problem solutions. Whatever the fundamental factors truly are, however, they need to be determined not only to aid in selecting programmers, but also to develop programmer and user training programs.

4. It is suggested that error-detection probability may be a function of a base-line error detection rate and, therefore, that seeding errors unknown to the individual checking the material may increase the probability of detecting an unseeded error. This conjecture should be subjected to experimental tests to determine whether seeding improves performance, and, if so, the frequency and type of seed-errors that should be used.

5. The issue of the basis for the superior performance of example-solutions needs to be addressed and resolved. As indicated in Conclusion #17 above, the concept of writing program code for each combination of factors (or the use of aids to automatically transform combination-independent logic into combination - dependent logic) and the transforming of the code written into alternate logical forms for feedback to the user for approval needs to be tested in an experimental environment. There is a potential here for substantially increasing the correctness of computer programs if the superior, almost error-free performance with example-solutions can be transferred to writing program code.

6. Independent of the methods for improving performance in writing program code suggested in Item 5 above is the concept of a language that combines general statements written in program code with redundant example-solutions, not as part of a program test, but, instead, as part of the program development so that the example-solutions may be inductively transformed into alternative code. A pre-compiler would produce actual code from both sources. Potential performance with such a language would benefit from the best properties of user-written code and user-generated examples.

ACKNOWLEDGEMENTS

The author is grateful to Robert F. Comeau who contributed significantly to the development of the initial concepts and who wrote the system and analysis programs for the experiment. His interest and enthusiasm for the work contributed to the success of the project. The author is also grateful to Pamela Johnson who, while with us for only a short time, contributed positively by conducting Experiments #3 and #4 and by providing data analysis for those experiments. Further, the author is grateful to Mike Olshausen for dedication to his task of editing throughout all revisions. Finally, the author is grateful to Dr. John J. O'Hare and Dr. Martin Talcott for their support and interest in the effort.

REFERENCES

Connelly, E. M., Comeau, R. F., Johnson, P. Effect of automatic processing on specification of problem solutions for computer programs. (Technical Report 81-361). Performance Measurements Associates, March 1981. AD A108570

Sheppard, S. B., Kruesi, E., and Curtis, B. The effects of symbology and spatial arrangement on the comprehension of software specifications (TR-80-388200-2). Arlington, Virginia: General Electric, October 1980.

Winer, B. J. Statistical principles in experimental design (2nd ed. pp. 727-736). New York: McGraw-Hill, 1971.

DISTRIBUTION LIST

OSD

CAPT Paul R. Chatelier
Office of the Deputy Under Secretary
of Defense
OUSDRE (E&LS)
Pentagon, Room 3D129
Washington, D.C. 20301

Department of the Navy

Engineering Psychology Programs
Code 442
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217 (5 cys)

Electronics & Electromagnetics
Technology Programs
Code 250
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Communication & Computer Technology
Programs
Code 240
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Tactical Development & Evaluation
Support Programs
Code 230
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Department of the Navy

Manpower, Personnel & Training
Programs
Code 270
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Information Sciences Division
Code 433
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Physiology & Neuro Biology Programs
Code 441B
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Commanding Officer
ONREAST Office
ATTN: Dr. J. Lester
Barnes Building
495 Summer Street
Boston, MA 02210

Commanding Officer
ONRWEST Office
ATTN: Dr. E. Gloye
1030 East Green Street
Pasadena, CA 91106

Department of the Navy

Office of Naval Research
Scientific Liaison Group
American Embassy, Room A-407
APO San Francisco, CA 96503

Director
Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375

Dr. Michael Melich
Communications Sciences Division
Code 7500
Naval Research Laboratory
Washington, D.C. 20375

Dr. L. Chmura
Code 7592
Naval Research Laboratory
Washington, D.C. 20375

Dr. Robert G. Smith
Office of the Chief of Naval
Operations, OP987H
Personnel Logistics Plans
Washington, D.C. 20350

Naval Training Equipment Center
ATTN: Technical Library
Orlando, FL 32813

Human Factors Department
Code N-71
Naval Training Equipment Center
Orlando, FL 32813

CDR Norman F. Lane
Code N-7A
Naval Training Equipment Center
Orlando, FL 32813

Department of the Navy

Dr. Alfred F. Smode
Training Analysis and Evaluation
Group
Naval Training Equipment Center
Code TAEG
Orlando, FL 32813

Dr. R. J. K. Jacob
Code 7590
Naval Research Laboratory
Washington, D.C. 20375

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Dean of Research Administration
Naval Postgraduate School
Monterey, CA 93940

Mr. Warren Lewis
Human Engineering Branch
Code 8231
Naval Ocean Systems Center
San Diego, CA 92152

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

Dr. John Impagliazzo
Code 101
Naval Underwater Systems Center
Newport, RI 02840

Naval Material Command
NAVMAT 0722 - Rm. 508
800 North Quincy Street
Arlington, VA 22217

Department of the Navy

Commander
Naval Air Systems Command
Human Factors Programs
NAVAIR 340F
Washington, D.C. 20361

Commander
Naval Air Systems Command
Crew Station Design
NAVAIR 5313
Washington, D.C. 20361

Mr. Phillip Andrews
Naval Sea Systems Command
NAVSEA 0341
Washington, D.C. 20362

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 81323
Washington, D.C. 20360

Dr. George Moeller
Human Factors Engineering Branch
Submarine Medical Research Lab
Naval Submarine Base
Groton, CT 06340

Head
Aerospace Psychology Department
Code L5
Naval Aerospace Medical Research Lab
Pensacola, FL 32508

Commanding Officer
Naval Health Research Center
San Diego, CA 92152

Dr. James McGrath
CINCLANT FLT HQS
Code 04E1
Norfolk, VA 23511

Department of the Navy

Navy Personnel Research &
Development Center
Planning & Appraisal Division
San Diego, CA 92152

Dr. Robert Blanchard
Navy Personnel Research and
Development Center
Command and Support Systems
San Diego, CA 92152

Mr. Stephen Merriman
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Dr. Julie Hopson
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Mr. Jeffrey Grossman
Human Factors Branch
Code 3152
Naval Weapons Center
China Lake, CA 93555

Human Factors Engineering Branch
Code 1226
Pacific Missile Test Center
Point Mugu, CA 93042

Mr. J. Williams
Department of Environmental
Sciences
U.S. Naval Academy
Annapolis, MD 21402

Dean of the Academic Departments
U.S. Naval Academy
Annapolis, MD 21402

Department of the Navy

Dr. S. Schiflett
Human Factors Section
Systems Engineering Test
Directorate
U.S. Naval Air Test Center
Patuxent River, MD 20670

CDR C. Hutchins
Code 55
Naval Postgraduate School
Monterey, CA 93940

Department of the Army

Mr. J. Barber
HQS, Department of the Army
DAPE-MBR
Washington, D.C. 20310

Technical Director
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Director, Organizations and
Systems Research Laboratory
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Technical Director
U.S. Army Human Engineering Labs
Aberdeen Proving Ground, MD 21005

Department of the Air Force

U.S. Air Force Office of Scientific
Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, D.C. 20332

Department of the Air Force

Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Dr. Earl Alluisi
Chief Scientist
AFHRL/CCN
Brooks AFB, TX 78235

Foreign Addressees

Dr. Kenneth Gardner
Applied Psychology Unit
Admiralty Marine Technology
Establishment
Teddington, Middlesex TW11 OLN
England

Director, Human Factors Wing
Defense & Civil Institute of
Environmental Medicine
Post Office Box 2000
Downsview, Ontario M3M 3B9
Canada

Dr. A. D. Baddeley
Director, Applied Psychology Unit
Medical Research Council
15 Chaucer Road
Cambridge, MA CB2 2EF
England

Other Government Agencies

Defense Technical Information Center
Cameron Station, Bldg. 5
Alexandria, VA 22314 (12 cys)

Other Government Agencies

Dr. Craig Fields
Director, System Sciences Office
Defense Advanced Research Projects Agency
1400 Wilson Blvd
Arlington, VA 22209

Dr. M. Montemerlo
Human Factors & Simulation
Technology, RTE-6
NASA HQS
Washington, D.C. 20546

Other Organizations

Dr. H. McI. Parsons
Human Resources Research Office
300 N. Washington Street
Alexandria, VA 22314

Dr. Jesse Orlansky
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

Dr. Robert T. Hennessy
NAS - National Research Council (COHF)
2101 Constitution Ave., N.W.
Washington, D.C. 20418

Dr. Robert Williges
Dept. of Industrial Engineering & OR
Virginia Polytechnic Institute & State University
130 Whittemore Hall
Blacksburg, VA 24061

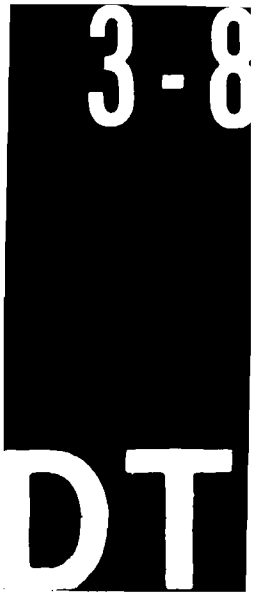
Dr. Deborah Boehm-Davis
General Electric Company
Information Systems Programs
1755 Jefferson Davis Highway
Arlington, VA 22202

Other Organizations

Dr. Edward R. Jones
Chief, Human Factors Engineering
McDonnell-Douglas Astronautics Co.
St. Louis Division
Box 516
St. Louis, MO 63166

Dr. Richard Pew
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02238

Dr. David J. Getty
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02238


3-8
DT